(2)

INTEGRATED VISION SYSTEM FOR A TARGET SEEKER

A FINAL REPORT

Submitted to the

Armament Directorate
Wright Laboratory
Eglin Air Force Base, Florida

and the

U. S. Army Strategic Defense Command, Huntsville, AL

by

Rafael M. Inigo
Principal Investigator
and
E. S. McVey
Co-Principal Investigator

Electrical Engineering Department
School of Engineering and Applied Science

University of Virginia
Charlottesville, Virginia

July 22, 1991

The following researchers collaborated on the research reported here and in the writing of this report.

Chewcharn Narathong, Ph.D., Consultant

Glenn Himes, Ph.D., Graduate Research Assistant

Chengho Hsin, MS

Jack Sigda, MS., Graduate Research Assistant

Gan Wang, Ph.D., Graduate Research Assistant

Qing Xu, Ph.D., Graduate Research Assistant

Begona Arrue, M.S., Graduate Research Assistant

Note: Dr. Himes was supported by a Virginia Engineering Foundation Fellowship and Ms. Arrue by a fellowship from the Spanish Ministry of Education.

PREFACE

This program was conducted by the Electrical Engineering Department, School of Engineering and Applied Science, University of Virginia, Charlottesville, Virginia 22903, under Contract No. DASG60-88-C-0030 with the U. S. Army Strategic Defense Command. Dr. John Johnson was the technical monitor and Ms. Starla Christakos and Mr. Jeff Brooks, WL/MNSI, were alternate technical monitors. The program was for a period of forty-two months, from February 21, 1988 to February 20, 1991. This is a final report covering the complete contract period.

This report is complemented by the following, more extensive and detailed, reports:

1. A008 Software test description

2. A009 Software test report: Simulations

3. A00A Engineering drawings, Level 1

4. A00B Subsystem design analysis report: Algorithms

5. A00C Final software manual

6. A00D Computer software product

TABLE OF CONTENTS

Section	Title	Page
I	**INTRODUCTION**	1
II	**MOTION/EDGE DETECTION**	5
	1. Introduction	5
	2. General Specifications	5
	3. The Modified Spatio-Temporal Algorithm	5
	4. Appliction of Simplifying Techniques to the Algorithm	9
	5. Simulations	14
III	**IMAGE SEGMENTATION**	15
	1. Introduction	15
	2. Segmentation Algorithm	15
	3. Problems	18
IV	**CENTROID DETERMINATION**	21
	1. Introduction	21
	2. Centroid Calculation	21
	3. Spatial Filtering for Peak Determination	23
	4. Hopfield Network for Peak Detection	25
	5. Simulation Results	27
V	**LOGARITHMIC SPIRAL IMAGING**	29
	1. Introduction	29
	2. Log-spiral Image Description	29
	3. A Hardware Implementation of Log-spiral Sensor	33
VI	**MULTI-PIXEL TARGET IDENTIFICATION SYSTEM**	38
	1. Introduction	38
	2. Position in the Integrated Vision system	38
	3. MHONN Design	38
	4. Simulations	46
	5. Summary	50
VII	**LINE CORRELATOR TRACKER FOR SCALING AND ROTATION**	52
	1. Introduction	52
	2. Line Correlator Tracker	52
	3. LCT Network Implementation	54
	4. LCT Simulations	58
VIII	**A SINGLE PIXEL TARGET DETECTION**	60
	1. Introduction	60
	2. The Pipeline Target Detection Algorithm	60
	3. Implementation Considerations	66

4. Simulations 66
5. Neural Network Implementation of the Pipeline System 68

IX **HOUGH TRANSFORM SINGLE PIXEL-TARGET DETECTION** 77
1. Introduction 77
2. Parallel Mapping Scheme 77
3. Peak Detection 79
4. Target Location 81
5. VLSI Implementation Analysis 81
6. Simulations 83

X **CONCLUSIONS AND RECOMMENDATIONS** 90
1. Conclusions 90
2. Recommendations 91

XI **REFERENCES** 93

LIST OF FIGURES

Figure	Title	Page
I.1	Multipixel System Block Diagram	3
II.1	Solution from Constant Equations	8
II.2	Block Diagram of the Motion/edge Detection	8
II.3	Filter Flow Structure	10
II.4	Approximation of Gaussian First Derivative	10
II.5	Filtering Hierarchies to form Gaussian Images	11
II.6	Simulation of Motion Detection algorithm: Synthetic Image	12
II.7	simulation of Motion Detection Algorithm: Real Image	13
III.1	Results for Robot Sequence	16
III.2	Segmented subimages	17
III.3	A Rotating rectangle	18
III.4	Objects in Two Overlapping Windows	20
IV.1	Missile Plume Images	22
IV.2	Centroids for Gray, level/Binary Images	22
IV.3	Network to Find Column Location of Centroid	23
IV.4	Parallel Analog convolution Network	25
IV.5	Hopfield Network	27
V.1	Log-Spiral Image Tessellation	30
V.2	Original Image of & Log-spiral Image	32
V.3	Analog Pixel-summing Arrangement	34
V.4	Digital Pixel-summing Arrangement	35
VI.1	Original Second-order Neural Network Architecture	39
VI.2	Architecture of the Modified Second-order Neural Network with Translational Properties	40
VI.3	Cydic Shift	43
VI.4	MHONN with Associative recall Learning Mechanism	46
VII.1	Row Search Mechanisms	54
VII.2	General Architecture of the Neural Network	55
VII.3	ICBM Plume Images	58
VIII.1	The Window-centroid Method	62
VIII.2	AND-Pipe Operations	62
VIII.3	Operation Flow of the PTDS Alorithm	64
VIII.4	A-AND Function	64
VIII.5	Simulation Results by the Original System Architecture with Salt-noise Sequences	67
VIII.6	Simulation results by the Original System Architecture with	

	Additive Gaussian Noise	69
VIII.7	BPN-implemented PE Structure	71
VIII.8	Straight-line Pattern Vectors	73
VIII.9	Examples of Negative Class Patterns	74
VIII.10	Training Pattern Examples	75
VIII.11	Simulation Results by the BPN-implemented System Architecture with Salt-noise Sequences	76
XI.1	Mapping Scheme	78
XI.2	Line Scheme	79
XI.3	Parameter Plane Summer with Thresholding	82
IX.4	Synthetic Image with Five Lines	84

LIST OF TABLES

Table	Title	Page
VI.1	MHONN OS, CS Invariant Pattern Recognition with Gray-level Inputs	47
VI.2	MHONN OS, CS Invariant Pattern Recognition with Binary Inputs	48
VI.3	MHONN Noise Tolerant Test Result - LSM icbm1 as input	48
IV.4(a)	MHONN Performance Test Result - Occlusion in LSM icbm1	50
VI.4(b)	MHONN Performance Test Result - Occlusion in LSM icbm3	51
VII.1	LCT Translation results for 90 Degree Plume	58
VII.2	LCT Rotation/Scaling results for 90 Degree Plume	59
VIII.1	Hardware Requirements for an NxN TP	75
IX.1	Simulation Results of Hough Transform	85
IX.2	Simulation Results of Peak Detection Scheme	86
IX.3	Simulation Results of Peak Detection Scheme	87
IX.4	Simulation Results of Peak Detection Scheme	88

GLOSSARY OF TERMINOLOGY

Activation Function -- the function that determines how the various input functions for the input classes are combined to produce a value that is used as the input to the transfer function.

Afferent --(1) forward connections in ANNs. Also called feedforward connections. (2) neurons that receive inputs from the external world in biological systems.

Analog-AND -- a two-operant AND function, one of the operants is a grey-level function and the other is a binary value.

AND-Pipe -- an array processor of the pipeline system implementing the Continuity Filter Algorithm.

Arc-of-Rings -- an approximation to the log spiral mapping that uses concentric circles and radial lines to define the log-polar geometry. This gives image elements that are shaped like arc sections of rings.

Array Processor -- An array of SIMD processing elements using conventional (nonassociative) random-access memory.

Artificial Neural Network -- (ANN) a computational geometry or computing structure that is designed to emulate some properties of real physical systems of neurons in some way. An ANN is defined by its geometry or structure which consists of elements and connections.

Association -- relating one piece of information to another or others. In some cases many events may associate to a single event. This may be a mapping of some events to others, or other relational transformation.

Associative Memory -- a particular type of adaptive filter operation.

Back Propagation Network -- the most popular ANN architecture. It learns by comparing the actual and desired outputs and modifying the weights in terms of the error.

Bimodal -- having values grouped into largely disjoint sets.

Cell -- (1) a neuron in a real neural system or (2) an element in an ANN.

Cell Plane -- a subset of elements in a slab that share the same receptive field shape. The centers of the fields may be at different locations.

Centroid - the center of mass of an object. For image processing, the point masses correspond to the pixel intensities.

Centroid Tracker -- a track initiation algorithm that estimates target trajectories based on the

trajectory continuity, constituting part of the PTDS algorithm.

Cluster -- a distinct, disjoint grouping of values that have some type of similarity, such as being within a certain range or boundary.

Collective Property -- a property of a system that occurs due to the largenumber of elements used. Alternatively called an emergent property.

Column -- a cut or grouping of elements across several layers. Represents a single signal pathway in the system., a depth cut or core of the network.

Comparator -- an analog integrated circuit which compares an input voltage to a threshold and forces the output to the voltage limits. For example, a 0 V to 5 V comparator with a 2.5 V threshold would force voltages smaller than 2.5 V to an output of 0 V and would force voltages larger than 2.5 V to an output of 5 V.

Competition -- selection of an element or elements based on some performance criterion, such as Euclidean distance from an input pattern.

Computation Plane -- the mapped version of the image plane under the LSM. The horizontal axis corrsponds to the natural log of the distance from the optical axis and the vertical axis to the angle. This is analogous to the representation of visual infromation in area 17 of the cortex in mammals.

Connectable area -- the area from which an element receives input connections. Can be sensory (as in the retina), or signal (as in subsequent neural layers) in nature. Also called the receptive field.

Connection -- unidirectional signal paths between elements. This is analogous to the axon in neural systems. Also called interconnections.

Continuity Filter -- a target tracking algorithm that enhances signal-to-noise ratio based on the target trajectory continuity, constituting part of the PTDS algorithm.

Current-Position Buffer -- a memory buffer at the Test-Pipe output that stores the tracking results of the most recent frame cycle.

Cyclic Shift -- a shift in the computation plane of the LSM due to object rotation.

Data Plane -- a memory organization referring to a memory space storing the pixel values of an entire image frame.

Efferent -- (1) backward connections in ANNs. (2) neurons that sent outputs to the external world.

Element -- simple Processing Element (PE), the basic building block of neural processors. Receives connections with their associated weights from external inputs and other elements,

combines them subject to some constrained functions, and produces a single output according to a transfer function. This is analogous to the neuron body itself in neural systems. Also called a cell.

Element Functions -- the functions that describe the operation of an element in a neural network, specifically, the input function, the activation function, and the transfer function.

Energy Function -- a function that describes the solution "surface" of a problem, usually derived through some type of Liapunov analysis. The solutions to the problem are the minima of energy function and the optimal solution is the global minimum.

Excitatory Cell -- a cell that receives only excitatory connections.

Excitatory Connection -- a connection with a positive (+) weight.

Expected False Alarm Rate -- statistically estimated false alarm rate based on the probability of error at a pixel location.

False Alarm Rate -- the value of accumulated false alarms averaged over the entire image sequence considered.

Feature -- some easily identifiable part of an object. Examples are edges, orientations of different parts, and things of this type. Motion can also be considered a feature.

Feedback Connections -- where every element in a slab is connected to every other element on that slab.

Form Invariance -- under some transformation the shape of the object will not change, i.e. the features remain the same. For example, form invariance under rotation means that shape formed is the same regardless of the angular orientation of the object.

Fully Connected -- where every element on a slab or layer is connected to every element on another slab or layer. Also called completely connected.

Functional Layer -- a group of elements within a layer that is used with other related function layers to generate the more complex functions of a conceptual layer.

Geometry -- the actual physical structure of an ANN. Describes the connections between elements.

Graded Learning -- learning in which the weights are modified according to a grade of the network's performance. Also called weakly supervised learning.

Grey level -- a black and white (shades of grey) representation of image intensity. Typically this is expressed as integers between 0 (black) and 255 (white).

Higher Order Neural Network -- a neural network capable of using double and triple products

of weighted inputs, in addition to linear combinations of these, in order to produce the desired output.

Highly Supervised -- learning in which the desired output must be known so that an error vector can be generated that is used in modifying the network's weights.

Histogram -- a plot of the number of pixels (vertical axis) in an image that have a certain grey level, for every grey level (horizontal axis).

Hopfield Network -- an artificial neural network capable of solving optimization problems, including the peak detection problem.

Hough Transform -- an image transformation technique devised for straight line determination in an image. Can be extended to higher order curves.

Image Plane -- the actual sensory representation of the image information, specifically in the LSM. Corresponds to the retinal arrangement of receptive fields in the retina.

Inhibitory Cell -- a cell that receives only inhibitory connections.

Inhibitory Connection -- a connection with a negative (-) weight.

Input Class -- a subdivision of the inputs to an element that contains inputs of the same type, i.e. all the elements of a class have the same data type, the same kinds of weights, and the same input function.

Input Function -- element function that describes how the inputs and weights of a certain input class will be combined to produce an input to the activation function.

Interconnections -- (1) see connections, and (2) inputs to an element from other elements, excluding external inputs.

Large-Scale -- the case where an image contains an object of many pixels so that the shape is recognizable.

Lateral Connection -- "side-to-side" connections between elements in the same layer or slab that allow such functions as competition and informational sharing.

Layer -- a specialized type of slab that receives essentially all of its inputs from a previous layer and sends signals to a subsequent layer. This suggests a temporal as well as functional grouping of cells.

Learning -- when a network has its connections or weights modified according to some rule. Modification of synapse strengths in biological systems.

Learning Rule -- a rule or algorithm that describes the functions and procedure used in the adaptation of a network's weights.

Log-Spiral-Mapping -- (LSM) a complex log of conformal mapping that approximates the mapping between the periphery of the retina (image plane) and the cortex (computational plane).

Mask Frame -- a two-dimensional binary array of the same size as the input image frames used by the Continuity Filter.

Mask Register -- an one-bit register in AND-Pipe implementing the Mask Frame.

Modified Higher-Order Neural Network -- a second order neural network with a modified architecture that reduces the number of weights by orders of magnitude.

Multiplexor -- a logic element which specifies the location in digital word form of a non-zero input line.

Neighborhood -- (1) a small group of elements around an element, and (2) a small group of pixels around a pixel. Also called a vicinity area.

Neural Network -- (1) see artificial neural network or (2) an actual system of neurons.

Neuron -- the building blocks of biological neural networks. Analogous to elements in ANNs.

Nonrecursive Linear Filter -- a digital linear filter whose unit sample response is zero outside of some finite area.

Optical Axis -- a coordinate system with its origin in the center of the image and its "horizontal" axis oriented according to a (possibly arbitrary) horizon in the original frame of reference. The vertical axis is orthogonal to this horizon in the same plane.

Optical Trajectory -- trajectory in the sensor caused by relative motion.

Ordinary Shift -- a shift in the interpretation plane of the LSM due to object scalings.

Paradigm -- a set of ideas that describes how a particular neural network operates. It contains a fairly detailed description of the important structural and functional issues, specifically the geometry, element functions, and learning rules.

Pattern Recognition -- (1) recognizing an object by its shape or (2) identification of a specific thing based on some characteristic features of that thing. Definition (1) is the most common in this report.

Pipeline -- a cascade of array processors.

Pipeline Target Detection System -- a real-time target detection and tracking system for single-pixel target detection and identification, a subsystem of the Integrated Vision System for a Target Seeker.

Pitch -- deviation of target from vertical plane.

Pixel -- short for picture element, (1) a single tiny piece of an image or (2) the light sensitive element that detects the image.

Plume -- the image produced by a missile during the boost phase of its flight. It is a large and bright object consisting primarily of the burning propellant and exhaust of the missile.

Receptive Field -- the area of a scene that excites the image plane or sensor.

Rectangular Grid -- rectangular image plane.

Region -- a subset of a slab in which all the elements share some common feature, such as receiving inputs from the same element or other region.

Reinforcement -- (1) the modification of weights, (2) a signal indicating the network's performance, and (3) a value used in weight modification.

Segmentation -- the partitioning of an image obtained by gathering its elements into sets likely to correspond to meaningful objects.

Self-Organization -- learning or synapse modification that takes place without a teacher of any sort.

Sigmoid -- a function often used in ANN's as an input-output relationship or transfer function for an element. A smooth function that asymptotically approaches two limits with a short of quasilinear region around some value as a transition between the two limits.

Signal-to-Noise Ratio (Mean Square) -- the mean square value of the signal to noise ratio.

Signal-to-Noise Ratio (Root-mean-square) -- the square root of the signal to noise ratio.

Small-Scale -- the case where an image contains an object that contains only a very few pixels so that the shap is not recognizable.

Spatial Filter -- an analog parallel network which convolves an input data set by the conductance values of the connections.

Target-Frame Buffer -- a memory buffer at the Test-Pipe output that accumulatively stores the reconstructed target trajectories.

Target Tracking -- altering the trajectory of the vehicle (or the orientation of the sensor) so that the object remains in a known fixed position on the sensor.

Threshold -- a value or decision boundary where all values below are set to a certain value all values above are set to a different value.

Time Constant -- a time value associated with each element or group of elements that

represents the processing time of the element. Analogous to a gate delay in a digital system.

Transfer Function -- actual input-output relationship in an element. Relates the value of the activation function to the output.

Translation Invariance -- a property of a system where the position of the object or information does not effect the operation of the system.

Translation, Rotation, and Scaling Invariant -- a mapping or ANN that recognizes patterns of the same form irrespective of their position, orientation and size.

Uniform-uniform Noise -- noise of uniform amplitude distributed uniformly.

Unsupervised -- learning in which the network modifies its weights by itself, without any supervision. The network learns correlations between items of information.

View Angle -- the angle, measured from the optical axis, between the vehicle's path and that of the target.

Visual Flight Control -- using visual information to stabilize a vehicle, control its path, and navigate to a specified goal.

Weakly supervised -- learning where the network updates its weights according to a grade of its performance. This is also called graded learning.

Weight -- the scaling factor of a connection to an element. In biological systems, this is called a synapse strength. The weights of a neural system constitute the stored information.

LIST OF SYMBOLS

SYMBOL	DEFINITION
A_k	Intensity mass in the Centroid Tracker algorithm
A_T	Intensity mass threshold in the Centroid Tracker algorithm
A_w	Detector area for the Continuity Filter
a_j	Activation function of a neural unit in layer j
$a_l(k,v,j)$	A weight for layer l, position v, connecting planes k and j
a_I	acceleration of ICBM
$\mathbf{a}_I$	acceleration vector of ICBM
$\mathbf{a}_{kew}$	acceleration vector of KEW
$\mathbf{a}_{rel}$	relative acceleration vector of ICBM w.r.t. KEW
a(x)	spatial smoothing function, x axis
a_i	variable weight for the connectable area input
A	angular position of ICBM as viewed from sensor
AoA	angle of attack of the KEW towards the ICBM
$\mathbf{a}_T$	Target acceleration in target motion analysis
A_l	weight connectable area of a U_s cell in layer l
ANN	artificial neural network
ATR	automatic target recognition system
A_V,A_v	gain of an amplifier
A, B, C'	Hopfield network constants
A-AND	Analog-AND function in the Continuity Filter theory
ALU	Arithmetic-logic unit
AP	AND-Pipe
B(k, ω)	Power spectral density of the background
BPN	Back propagation network
BSTS	Boost surveillance and tracking system
$b_l(j)$	B weight in layer l of plane j
C	one-dimensional correlation function

C_i	capacitance of node i
$c_{l-1}(v)$	c weight for layer l-1, position v
CF	Continuity Filter
CPB	Current-Position Buffer
CT	Centroid Tracker
CU	Control Unit
$d_l(v)$	d weight for layer l, position v
$d_{lmax}(v)$	maximum **d** weight value for a connectable area
D	maximum horizontal displacement
D_l	D weight connectable area of a U_c cell in layer l
D	initial distance betwween KEW and ICBM
D	Longer dimension of target in imaging analysis
$D_k(s)$	Time difference of image intensity in differencing methods
d(**r**)	Detector function for the Continuity Filter
d_k	Decision function in centroid computation; desired output signal of the back propagation network
$E_{\min}$	minimum energy for a Hopfield network
E(x,y,t)	intensity at point (x,y) and time t
$e_{i,j}$	intensity of pixel at location i,j
$e_{i,j,k}$	k-direction neighborg of $e_{i,j}$
$\bar{e}$	average excitatory value
$e_{\max}(k, \mathbf{n})$	excitatory value of a cell in plane k, position **n**
$e_{\max}(k, \mathbf{n})$	maximum possible excitatory value of a cell
E_{alfa}	partial derivarive of E with respect to *alfa*
EFAR	Expected false alarm rate
F(O)	fourier transform evaluated at O
f(x,y,t)	an input image at time t
$f_j(a_j)$	Transfer function of a neural unit in layer j
FAR	False alarm rate
FIFO	First-in-first-out
FOV	Field of view
G_{SNR}	Signal-to-noise ratio gain

$H(\mathbf{k}, \omega)$	Transfer function of optimum filter
H_o	maximum value of spatial filter
$h(\mathbf{n})$	inhibitory value of a cell at position **n**
HT	Hough transform
HTS	Hough transform system
I	Image coordinate system in target motion analysis
I_i	Image intensity in image formation
$I_k(s)$	Image intensity at frame k in differencing methods
I_s	Spectral intensity of a point source in image formation
I	electric current
I_i	inputh current to i^{th} Hopfield node
I_i	inputh current to i^{th} Hopfield node
I_{ro}	current leaving Hopfield node through resistor *ro*
I_C	current leaving Hopfield node through capacitor
I_o	current leaving amplifier of Hopfield node
I_{peak}	peak current into a Hopfield network
$I_{peak}(min)$	minimum current into Hopfield network to turn node on
$I_{nonpeak}(max)$	maximum current into Hopfield net which leaves node off
I(x,y,t)	3-D intensity function in rectangular format
$I(r, \theta, t)$	3-D intensity function in polar form
I(u,v,t)	3-D intensity function in the computation plane
$\hat{I}_k(s)$	Shifted image intensity at frame k in differencing methods
ICBM	Inter-continental ballistic missile
IFOV	Instantaneous field of view
IVSTS	Integrated Vision System for a Target Seeker
K_{cl}	number of U_c planes in layer l
K_{sl}	number of U_s planes in layer l
KEW	Kinetic energy weapon
LSM	log-spiral mapping
L_θ	second order oriented smooothing function
M_T	total intensity mass of an object

m_o	zero order moment
m_1	first order moment
m_k	mean location of an object on k axis
m_{ij}	in tensity of image pixel (i,j)
MF	Mask Frame
MPP	Massively parallel processor
MR	Mask Register
MTI	Moving target indication
MTT	Multiple target tracking
L	Initial target distance from the sensor in imaging analysis
N	number of pixels in a row or column of a square sensor
N_s	Number of frames in an image sequence
n(**r**, t)	Three-dimensional noise signal
n	row and column coordinates of a neocognitron cell within a cell plane
o_i	Output signal of a neural unit in layer i
p(v)	testing pattern for P(v)
P(v)	training pattern for a neocognitron
P_o	total object power
P_e	Probability of error
$P_{e\|b}$	Upper bound of probability of error
$P_{e\|s}$	Probability of error for signal
$P_{e\|n}$	Probability of error for noise
$\mathbf{P}_n$	North pole in vision system geometry
P_n	Probability of a binomial noise distribution
$\mathbf{P}_s$	South pole in vision system geometry
P_s	Probability of a binomial signal distribution
$\mathbf{p}_i$	Image point in the image coordinate system in image formation
$p_n(\mathbf{r})$	Average power of noise n(**r**, t)
$p_s(\mathbf{r})$	Average power of signal s(**r**, t)
p	Homogeneous coordinates of the image point **p** in image formation
PDP	Parallel distributed processing

PE	Processing element
PTDS	Pipeline Target Detection System
q_l	reinforcement scale factor for layer l
r	radius, in units of length
{R}	set containing discrete values of the intensity at t_1
R_{ij}	Hopfield resistive connection between nodes i and j
r_l	inhibition constant for layer l
r_{lmax}	maximum possible inhibition constant which allows recognition
$R_n(\omega)$	Autocorrelation function of noise distribution
$\mathbf{r}_T$	Target location in image plane
$\mathbf{r}_0$	Initial target location in image plane
RMTI	Recursive moving-target-indication
$S^*(\mathbf{k},\omega)$	Complex conjugate of 3D signal Fourier transform
s	measure of directional angle between two neocognitro vectors
s(r, t)	Three-dimensional target signal
s_d(r, t)	Three-dimensional filtered target signal
SIMD	Single instruction stream multiple data stream
SNR	Signal-to-noise ratio
$(SNR)_o$	Signal-to-noise ratio of the original image signal
$(SNR)_d$	Signal-to-noise ratio of the filtered image signal
$(SNR)_P$	Signal-to-noise ratio in a processing state P
SPTDT	Single-pixel target detection and tracking
T	Intensity threshold
T_{ij}	Hopfield weight connection between nodes i and j
t_1	time corresponding to the first frame
t_2	time corresponding to the scond frame
{T}	set containing discrete values of intensity at *tsub* 2
T_{ij}	transconductance from neuron i to neuron j
t_k	Training pattern
TAS	Target association system
TDT	Target detection and tracking
TFB	Target-Frame Buffer

TP	Test-Pipe
TWC	Temporal window column
u	x component of velocity
U_i	sum of currents at input node i (Hopfield)
$\hat{\mathbf{u}}$	unit vector in 3D space (x,y,t)
$U_{cl}(k, \mathbf{n})$	output of excitatory U_c cell, layer l, plane k, position **n**
$U_{sl}9k, \mathbf{n})$	output of excitatory U_s cell, layer l, plane k, position **n**
U_o	output of cells in input layer of neocognitron
u_i	input voltage of ith Hopfield node
$u_i(min)$	minimum possible input voltage for Hopfield node
$u_i(max)$	maximum possible input volatge for Hopfield node
u_H	highest input voltage for linear op amp operation
u_L	lowest input voltage for linear op amp operation
v	velocity vector
$\mathbf{v}_I$	Velocity of inter-continental ballistic missile
$\mathbf{v}_K$	Velocity of kinetic energy weapon
$\mathbf{v}_r$	Radial component of relative velocity in target motion analysis
$\mathbf{v}_S$	Sensor velocity in target motion analysis
$\mathbf{v}_T$	Target velocity in target motion analysis
$\mathbf{v}_t$	Tangential component of relative target velocity
$\mathbf{v}_w$	Object point in the world coordinate system in image formation
$\mathbf{v}$	homogeneous coordinates of an object point
V	state of a neuron in LCT Hopfield network
V_H	output voltage for a logical 1, Hopfield net
V_L	output voltage for a logical zero, Hopfield net
V_i	output voltage of ith Hopfield node
$V_{\max}$	maximum output voltage of Hopfield node
V_t	threshold voltage for operational amplifier
V_x	operational amplifier input voltage
v	y component of velocity
W	World coordinate system in target motion analysis
w(m, n)	Two-dimensional window function

w_{ji}	Connection weight between layers i and j of a feed-forward network
$\bar{x}_f$	mean location of f(x)
$\hat{x}_k$	X coordinate of the intensity centroid in the Centroid Tracker
$\hat{y}_k$	Y coordinate of the intensity centroid in the Centroid Tracker
α	Step of the momentum term in weight adaptation
α	degree of saturation of the *Psi* function
η	normalization value of the spatial filter
η_i	centroid location on i axis
η_j	Centroid location on j axis
ΔX	overall horizontal estimate, LCT
ΔY	overall vertical estimate, LCT
Δx	true horizontal displacement
Δy	true vertical displacement
$\Delta\hat{x}$	individual horizontal estimate
$\Delta\hat{y}$	LCT individual vertical estimate
Δ_B	Subpixel shift estimate in differencing methods
$\Delta \mathbf{r}$	Spatial translation of target signal in image plane
$\delta(\cdot)$	Two-dimensional unit sample function
δ_j	Error signal at a neural unit in layer j
δ_{oi}	Error vector at the output layer
δ_{hj}	Error vector at the hidden layer
η	Learning rate in weight adaptation
$\Gamma_k^{D\times w^2}$	3D space of TWC, size $D\times w^2$ at x_k,y_k
∇^2	Laplacian operator
κ	scaling parameter
λ	Wavelength in image formation
ω_i	Solid angle originated from the image plane in image formation
ω_s	Solid angle originated from a point source in image formation
Φ	Sensor's total field of view in imaging analysis
$\Phi_n(\omega)$	Power spectral density of random noise
ϕ	Solid angle subtended by the longer dimension of the target

$\phi_i(\lambda)$	Spectral radiant flux emitted by the lens in image formation
$\phi_s(\lambda)$	Spectral radiant flux received by the lens in image formation
$\phi(x)$	Fukushima's output function for U_s cells
$\Psi(x)$	Fukushima's output function for U_c
π_s	*a priori* probability of signal
π_n	*a priori* probability of noise
Ψ	Receptive-field pixel set in BPN-implemented PE design
σ	standard deviation
σ_n	standard deviation on n estimates
σ_t	temporal standard deviation
$\sigma_{x,y}$	spatial standard deviation
θ_j	Internal bias of a neural unit in layer j

SECTION I

INTRODUCTION

This report summarizes the results of research on *An Integrated Vision System for a Target Seeker, (IVS)*. Detailed explanations of methods, results, simulations, software, etc. can be found in other final reports delivered (A008, A009, A00A, A00B).

Figures provided by WL/MNSI on system geometry and requirements are: Field of view of from 100 to 300 μ rad/sec of solid angle. the initial distance between weapon and target is of the order of 100 Km with a closing velocity of 6 to 10 Km/hr. The target dimensions can be between 1m wide-2m long and 3m wide-16m long. Simple geometric considerations show that the above figures are compatible with a feasible system.

It is required that the IVS performs in real time and, from the above figures, it is clear that the time from ICBM launching to impact with the intercepting weapon will be just a few seconds.

In view of the requirement for real time operation, it was decided that as many parts of the system as possible would be designed using artificial neural networks (ANNs). ANN research has experienced a big increase in activity in the last few years. Among the reasons for this, we can mention the ability of ANNs to recognize patterns even under severe degradation due to noise and other artifacts.

In spite of all the research activity on ANNs, progress on their analog VLSI implementation has been very slow. Among the reasons for this are: huge numbers of weights (resistors) and connections (and hence a serious routing problem), heat dissipation, etc. Several small scale ANNs have been made commercially available recently, [1,2]. The capabilities of these networks, however, are very limited, due to the small number of processing elements (neurons) and weights they contain. In the course of our research we have developed a Hopfield-Tank ANN for centroid determination and a modified higher order NN (MHONN) for pattern recognition. It is shown in the *Subsystem Design Report-Algorithms* that the first is implementable in analog VLSI, and the second in hybrid VLSI form for a sensor resolution of 128x128. We consider this to be an important contribution.

The task to be performed by the system consists of:

a) acquiring the image when the target (ICBM) is launched. At this point, due to the booster and, mainly, to the plume, the target has big dimensions and can be termed *multipixel*.

b) detecting target motion within the image

c) identifying the target (i.e., pattern recognition applied to target identification).

d) tracking the target (keeping it centered on the optical axis).

These steps refer to the *multipixel* case. After the post-booster stage, the target will become very small, of the order of one pixel, and we will then have to resort to identifying and tracking a *single-pixel* target. So, steps (c) and (d) will have to be "repeated" for the *single-pixel* case. When the weapon gets close to the target, the target image will, again, become *multipixel*. A simplified block diagram of the multipixel system is shown in Fig. I.1.

Each one of the six multipixel blocks will be described in more detail in Section II and following.

A summary of the operation of the multipixel system is as follows:

a) The image is acquired by a 128x128 infrared rectangular tesselation sensor. Block I, *Motion and Edge Detection* detects objects experiencing motion, determines their edges, and binarizes the resulting edge image. It must be noted that the original grey level image will be needed in block III.

b) Next, in Block II, *Segmentation*, objects are segmented, i.e., they are separated into windows, one per moving object. the segmentation algorithm takes into account the possibility of parts of an object falling within another object's window. the partially included object is deleted from the window and only a complete object remains in each window. Segmentation uses standard image processing techniques.

Notice in Fig. I.1 that once the objects are segmented, all further processing can be done for all the windows in parallel.

c) Block III, *Centroid Calculator*, is used to determine the centroid of the object in each window. In order to calculate the centroid coordinates in terms of its own window coordinates, an **analog parallel network** (APN) was designed. The APN performs a spatial filtering to generate a peak value corresponding to the centroid location. A Hopfield network then determines the peak location of the centroid. Both operations, filtering and peak location, are separable to their x (row) and y (column) components. Consequently, the complete centroid calculation can be done with two one-dimensional networks.

The target is identified by subsystem V, *see (e) below*. The target is identified, only the target window needs to be considered, and a continuous update of the centroid location is used for tracking. It must be noted that, although the window coordinates provided by the previous block are used for centroid calculation, the centroid is determined in terms of the grey level object, since for the objects of interest there is a substantial intensity variation from the brightest point to the darkest. If the binary image would be used, the centroid would not take this fact into account and subsequent steps would not produce correct results.

d) Each window is mapped to a log-spiral grid in Block IV, *Log-spiral-map* (LSM). This mapping can be performed in software, with special purpose digital circuitry, or with an ANN. A conformal transformation is used to map the log-spiral grid to a computation plane in which rotations on the optical axis and scalings are transformed to displacements along the two coordinate axes, respectively. This property of the transformation is used for two purposes: (i) to be able to recognize an object in different scales and orientations by means of a MHONN, Block V, and (ii) to be able to determine object rotation and orientation using the line correlator tracker (LCT) of Block VI in the computation plane of the LSM.

e) The next step in the process is to identify the window(s) that contains a valid target. This is done by Block V, MHONN. This is a second order MHONN which is able to recognize images that are translated in the plane. Since the LSM transforms scaling and rotations into translations, and the unmapped image is always centered on the centroid,

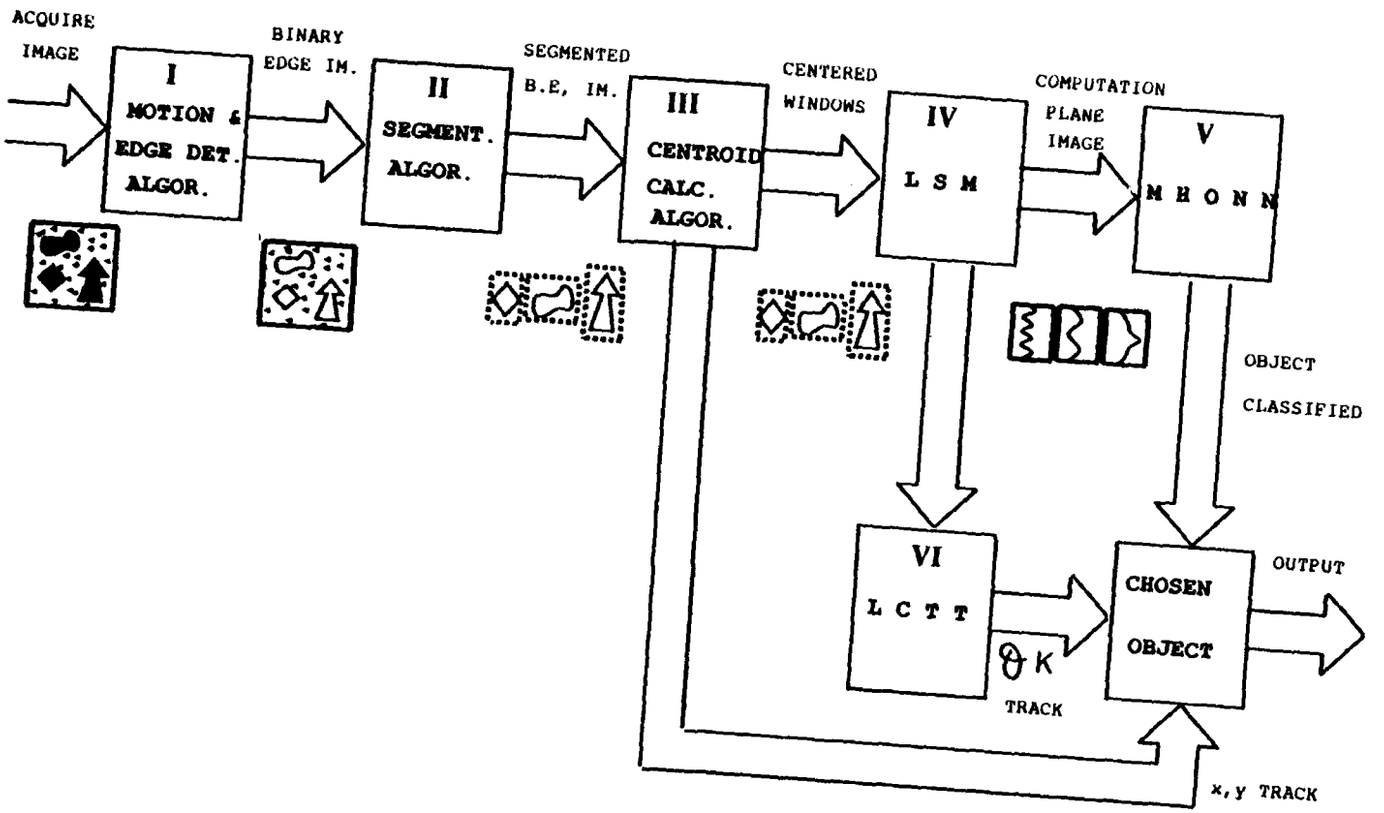


Figure I.1 **MULTIPIXEL SYSTEM BLOCK DIAGRAM**

the MHONN, working with images in the computation plane of the LSM, is able to recognize objects of different sizes and orientations.

f) Block VI, LCTT, is used to determine object scaling and rotation by computing translations in the LSM computation plane. This needs to be done only for the valid target window. As its name indicates, the LCTT is based on linear correlation, instead of the usual 2D correlation. An ANN Hopfield network has been designed for its implementation.

Once the object of interest has been identified, as mentioned above, it is tracked by computing its centroid from frame to frame. Its size and orientation are determined by means of the LCTT.

For the single pixel situation, the only information available is *target continuity* and target intensity invariance for short time intervals (the sensor is infrared and temperature does not change instantaneously).

In view of the limited information available, a novel filter, the *Pipeline Target Detection System* (PTDS) was developed to filter out noise and other disturbances and retain target pixels, which in a time sequence of several frames reproduce the target trajectory. the continuity filter consists of two main parts, the *and pipe* and the *test pipe*. In the and pipe, 3x3 masks of the image are ANDed, **analogically** with a mask of ones to detect presence or absence of candidate pixels. The test pipe than determines whether the candidate pixel (if it exists) can be a trajectory pixel, based on its change in spatial position from frame to frame. A further improvement of this, which produces better results, is the use of a backpropagation NN to track candidate pixels. All the parts of this subsystem can be implemented by means of massive parallel digital processors.

As mentioned above the output of the PTDS is an image containing target trajectories. At this point, the location of the target(s) at frame "n" in the sequence is not known. The coordinates of each of the last frame's pixels is known. Some of these pixel may correspond to valid trajectories and some to noise. By means of a Hough-transform implemented in ANNs, it is then determined which straight line segments (valid trajectories) are present in the image, and for these, the location of the last frame pixel gives target position. This network is implementable in analog VLSI.

SECTION II

MOTION/EDGE DETECTION

1. INTRODUCTION

Currently used algorithms for motion detection make some assumptions on the sequence of images used for this purpose which are not exactly true in most cases. The most common of these assumptions is the constancy of scene illumination, by which it is meant that any changes in intensity at a given image point must be due to object motion. This assumption is used in intensity-based gradient schemes. In feature-based gradient schemes, on the other hand, the aperture problem presents a serious challenge to the designer. Shortly stated, the aperture problem means that only the component of velocity perpendicular to an edge can be uniquely determined.

We have developed an algorithm and propose to use an original architecture implementable in real time, that determines instantaneous velocity of all objects present in the image sequence and detects their edges, producing as a result a binarized edge image and an indication on the direction of motion of each object. No simplifying assumptions are made about the image sequence, and the aperture problem is solved.

In what follows we will refer to "three dimensions" meaning dimensions x,y and t, that is, a time sequence of two dimensional images.

2. GENERAL SPECIFICATIONS

The following general specifications are satisfied by the motion-edge detection system:

a) A velocity field is estimated for the sequence. This involves solution of the aperture problem.

b) Velocity vectors accurately represent optical flow within small volumes dV = dxdydt. These vectors are calculated only at "prominent feature points," i.e., points at which an important image feature, such as intensity, experiences a significant change.

c) The algorithms is free of strong restrictions or assumptions on the scene contents or image formation process.

d) The algorithms is applicable to a wide variety of natural imagery and has high noise immunity.

e) The algorithms have a simple structure and are implementable in real time.

3. THE MODIFIED SPATIO-TEMPORAL ALGORITHM

Based on the above general specifications, a modified spatio-temporal filter was designed and tested by means of software simulations . The filter is "modified" in the sense that it does not use the assumption of constant intensity over time. In addition the aperture problem that arises in other schemes such as the feature-based gradient schemes, has been solved.

The basic processes involved in the algorithm are spatio-temporal filtering and velocity estimation. The first is achieved by means of oriented spatio-temporal filters. The velocities are then estimated by the gradient approach applied to the filtered image sequence. This approach smoothes the images, thus radically reducing the effect of noise and, in addition, direct computation of derivatives (i.e., gradients) is avoided, thus eliminating noise enhancement. This approach also permits motion detection of low contrast objects. Edge features, i.e., sharp changes in intensity, are basic features used by the algorithm. The zero crossing of second directional derivative is used to detect edge features because it is the most reliable method [1,2]. If the derivative is taken in different directions at a point, the zero crossing occurs at the same (edge) point for all the directions. Second zero crossing directional derivatives require gradient information in addition to second directional derivatives.

a. Algorithm Design

The image sequence f(x,y,t) is convoluted with a smoothing spatio-temporal function G(x,y,t),

$$T(x,y,t) = f(x,y,t) * G(x,y,t) \tag{II.1}$$

The second directional derivatives of T(x,y,t) at edge points with respect to a spatial vector s_i that may have any direction from 0 to 2π when computed for a spatio-temporal sequence at (x,y,t) and (x+dx, y+dy, t+dt) and equated at the two points in the sequence, is the basis for edge detection. Further processing provides velocity information at the edges solving, in the process, the aperture problem.

The algorithm is divided in two main parts: edge detection and motion (velocity) determination. The second partial derivative of T with respect to a directional vector $s = |s|(\cos\theta i + \sin\theta j)$, where θ can take on any value between zero and 2π, is computed at an edge point at times t and t+dt. Its value should be the same at both instants [3]. From the derivative rule for convolution,

$$K(x,y,t) = \frac{\partial^2 T(x,y,t)}{\partial s^2} \tag{II.2}$$

$$= f(x,y,t) * L_\theta$$

The vector s can have any direction from zero to 2π and is related to the x and y axes directions by the directional cosine and sine, respectively. Thus

$$L_\theta = \frac{\partial^2 G}{\partial s^2} = \frac{\partial^2 G}{\partial x^2}\cos^2\theta + 2\,\frac{\partial^2 G}{\partial x \partial y}\cos\theta\sin\theta + \frac{\partial^2 G}{\partial y^2}\sin^2\theta \tag{II.3}$$

Although different directions of the second derivative can be taken at a point, all their zero crossings occur at the same edge point. Thus, the zero values of equation (II.2) will correspond to edge points. Expanding K(x,y,t) by means of a Taylor series,

$$\frac{\partial K}{\partial x}u + \frac{\partial K}{\partial y}v + \frac{\partial K}{\partial t} = 0 \tag{II.4}$$

where u and v are the x and y components of velocity at point (x,y,t), respectively.

Two motion constraint equations are needed to obtain a solution for u and v, Fig. II.1. In practice we use

$$\left[f * \frac{\partial L_{\theta i}}{\partial x}\right] u + \left[f * \frac{\partial L_{\theta i}}{\partial y}\right] v + \left[f * \frac{\partial L_{\theta i}}{\partial t}\right] = 0; \; 0 \le \theta_i < 2\pi \qquad \text{(II.5)}$$

where

$$\frac{\partial L_\theta}{\partial j} = \frac{\partial^3 G}{\partial x^2 \partial j}\cos^2\theta + 2\frac{\partial^3 G}{\partial x \partial y \partial j}\cos\theta\sin\theta + \frac{\partial^3 G}{\partial y^2 \partial j}\sin^2\theta \qquad \text{(II.6)}$$

with j= x,y,t .

In (II.5) θ is given two or more convenient values (0,45°, 90°,...). The solution for u an v is obtained from (II.5), after computation of the bracketed terms using the LSE method and solving by a pseudo inverse matrix technique.

The spatio-temporal smoothing function G(x,y,t) is chosen as

$$G(x,y,t) = g(x,\sigma_x)g(y,\sigma_y)g(t,\sigma_t)$$

$$= \left[\frac{1}{\sqrt{2\pi\sigma_x}} e^{-\frac{x^2}{2\sigma^2 x}}\right]\left[\frac{1}{\sqrt{2\pi\sigma_y}} e^{-\frac{y^2}{2\sigma^2 y}}\right]\left[\frac{1}{\sqrt{2\pi\sigma_t}} e^{-\frac{t^2}{2\sigma^2 t}}\right] \qquad \text{(II.7)}$$

The values of σ_t, σ_x, σ_y determine the sensitivity to velocity. The maximum response is obtained for (with $\sigma_x = \sigma_y = \sigma$),

$$v_{opt} = \frac{\sigma}{\sqrt{2\sigma_t}} \qquad \text{(II.8)}$$

A range of velocities has to be detected, thus it is be necessary to have a multi-channel system in which the σ and θ parameters are selected for optimal detection. A block diagram of the system is given in Fig. II.2.

b. Number of Filtering Operations

In order to implement the edge/motion detection spatio-temporal filter, the operations described by equations (II.1) to (II.6) must be performed. A total of twelve filtering operations per channel, i.e. convolutions of G and its partial derivatives with the image function, must be implemented.

For the range of velocities of our particular application, σ_t=1 and σ_x=σ_y have values of 1, 2, 3, and 4. These were determined heuristically.

Convolution is performed, of course, in a discrete fashion, i.e., as a discrete summation of weighted signals intensities over the kernel.

b.1. Temporal function approximation

Convolution requires, in general, much more processing time than the solution of (II.5) for the velocities u and v . Considering only the filtering (or convolution) process, the total number of operations required is equal to the product of the number of filters

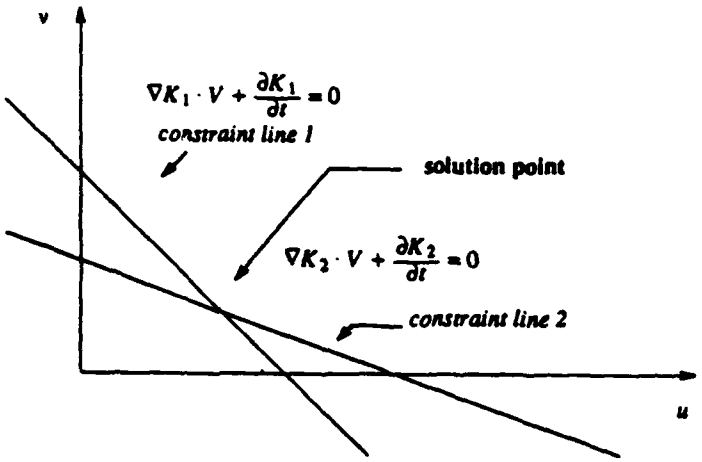


Figure II.1 A unique solution for (u,v) can be obtained from two motion constraint equations

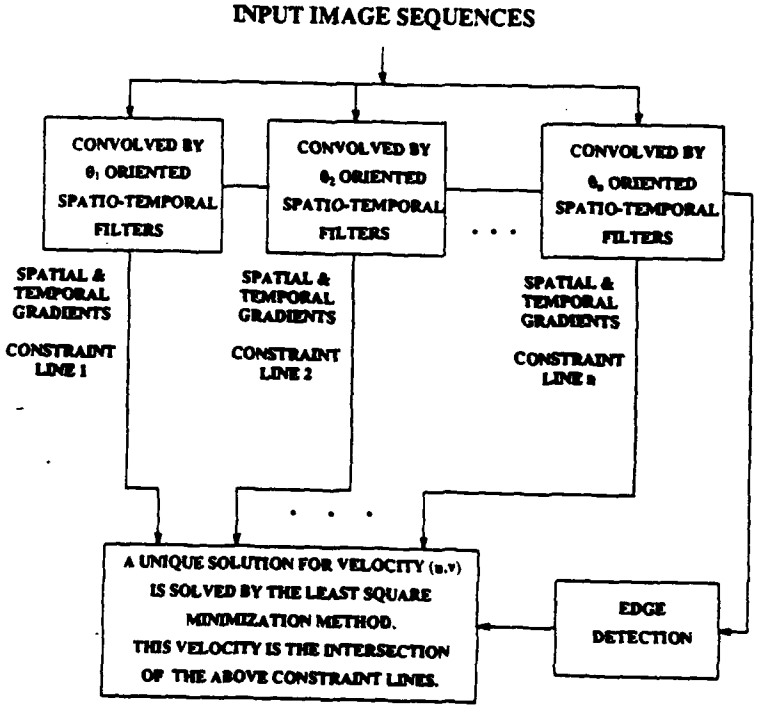


Figure II.2 Block diagram of the motion/edge detection filter

(twelve) times the sum of the kernel size for each of the four channels (3x nxn) times N^2 (where N is the image resolution.) For N=128 this is equal to 1.2 billion summations and multiplications , an impossible figure to deal with in real time.

The filtering operations are separable into three 1D filters. In that case, the spatial filter sizes are 9, 17, 25 and 33, with temporal filter size of three. Two temporal filter operations are required, and 19 spatial operations are required per channel. Thus, the number of computations required is

$$N.C. = (2x\,3+19x\,(9+17+25+33)xN^2 = 26\ \ 10^6$$

which, although a considerable reduction, is still too large for digital processor computation. If digital signal processing chips are used and about one million operations per second are possible, more than 26 seconds would be required for the filtering operations only.

b.2. Computational complexity reduction for digital implementation

A tree showing the required filtering operations for the 1D implementation is shown in Fig. II.3. The notation is as follows: jn means the nth partial derivative of G with respect to j ; thus x1 is dG/dx. The figure shows that the filter requires 2 temporal filtering operations and 19 spatial filtering operations. These operations must be performed for each of the four channels. A property of convolution of Gaussians is that if two Gaussians are convoluted, the results is also a Gaussian with a larger standard deviation than either. Thus, a cascade of Gaussians can be used as a hierarchical filter on the input image. A variation of this idea was developed in [4] and is applied here. The method is called "Hierarchical Discrete Correlation" (HDC). Recall that in addition of filtering with Gaussian of different σ, it is necessary to filter the image with 1st, 2nd and 3rd partial derivatives of G. We have developed a technique to combine Gaussians to form the 1st, 2nd and 3rd derivatives. In addition, convolution is distributive, i.e., (f1 + f2) * F = f1*F + f2*F, so Gaussian smoothed images can be combined to approximate desired filtering functions. The method used to approximate the first derivative is illustrated in Fig. II.4. Similar techniques are used to obtain the second and third derivatives. A third approximation useful in reducing computation uses subsampling, i.e., calculating the filtered image at less pixel locations than the original image resolution.

Since high frequency components are filtered out by the Gaussian filtering operation, the highest frequency component of the filtered image is lower. Thus, the filtered image does not need to be sampled as finely as the original image.

4. APPLICATION OF SIMPLIFYING TECHNIQUES TO THE ALGORITHM

The first technique to be applied is the formation of a set of Gaussian smoothed images using HDC. Figure II.5 illustrates this method. Two hierarchies are used to efficiently generate a set of Gaussian smoothed images in which the standard deviation varies by the square root of two.

The next step is to combine linearly these Gaussian smoothed images to form the desired first, second, and third derivative of Gaussian filtered images. This is a heuristic approximation process that produces good results.

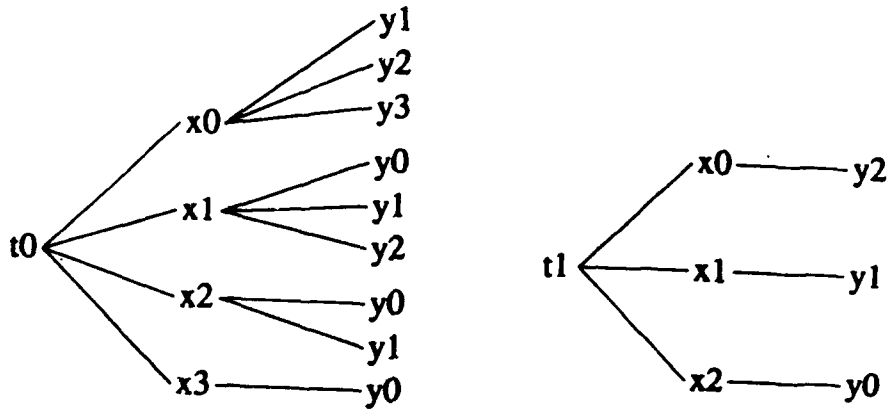
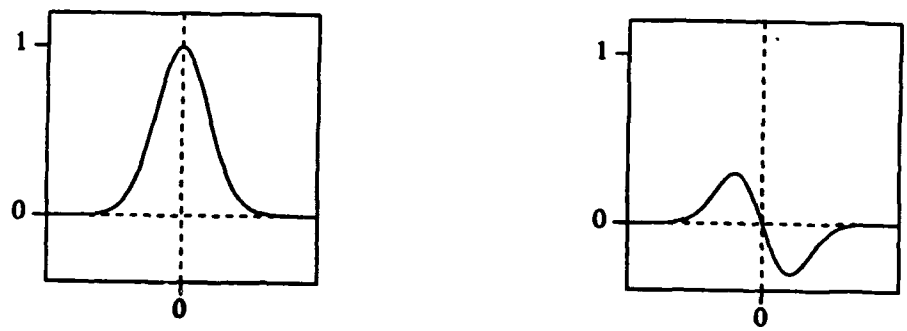


Figure II.3 Filter flow structure



Gaussian, $G_\sigma(x) = e^{-x^2/2\sigma^2}$

1st Derivative of Gaussian

Graphs of Gaussian and Derivative of Gaussian Functions

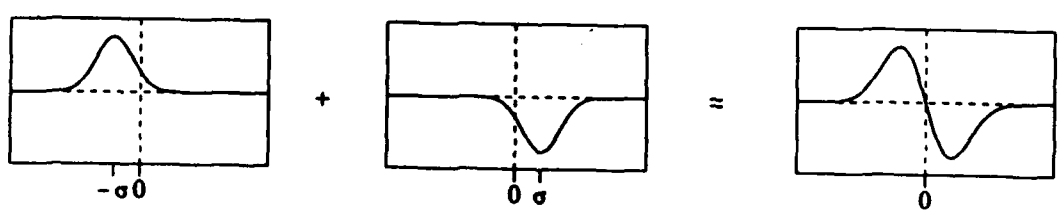
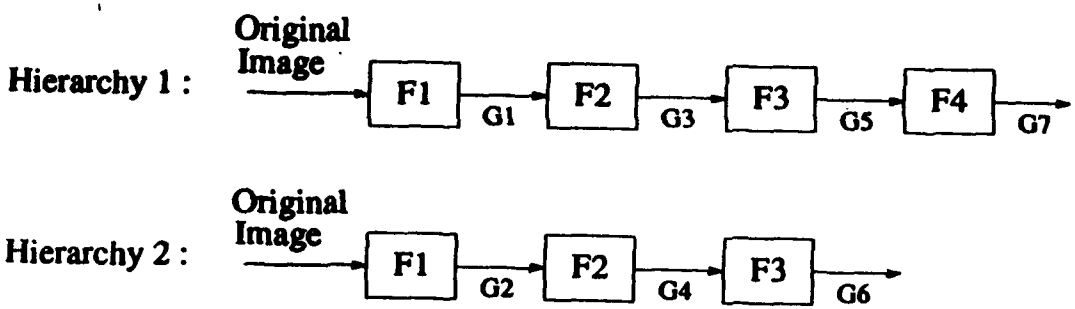


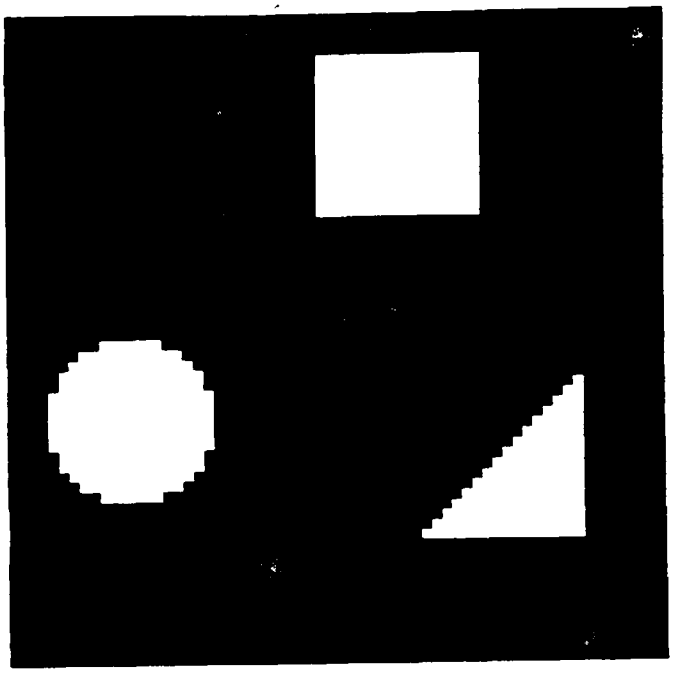
Figure II.4 Approximation of Gaussian first derivative



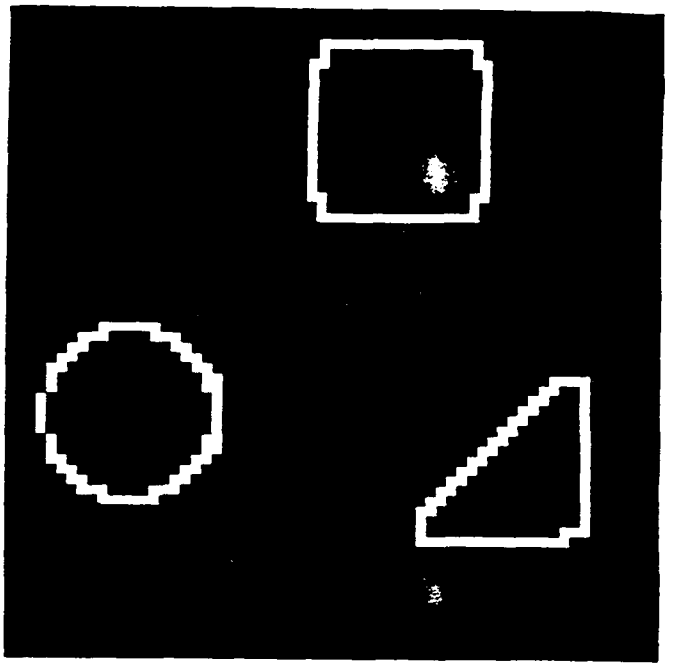
- Hierarchy 1 : Kernel size = 5;
  F1 - Gaussian filter, sigma = 0.707;

- Hierarchy 2 : Kernel size = 7;
  F1 - Gaussian filter, sigma = 1.00;

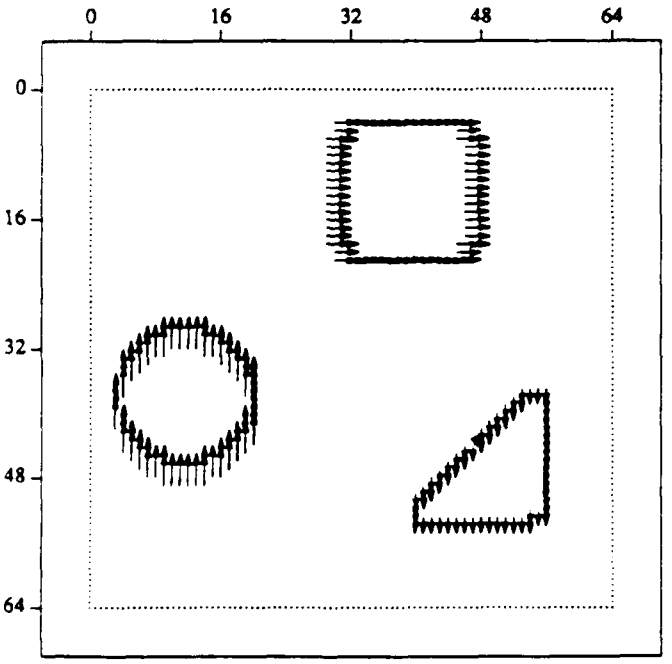
Figure II.5 Filtering hierarchies to form Gaussian smoothed images



original image

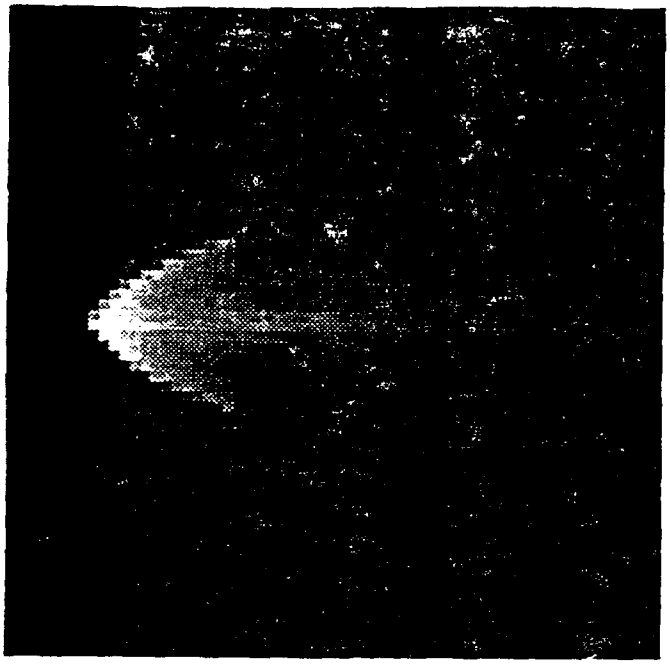


edge image

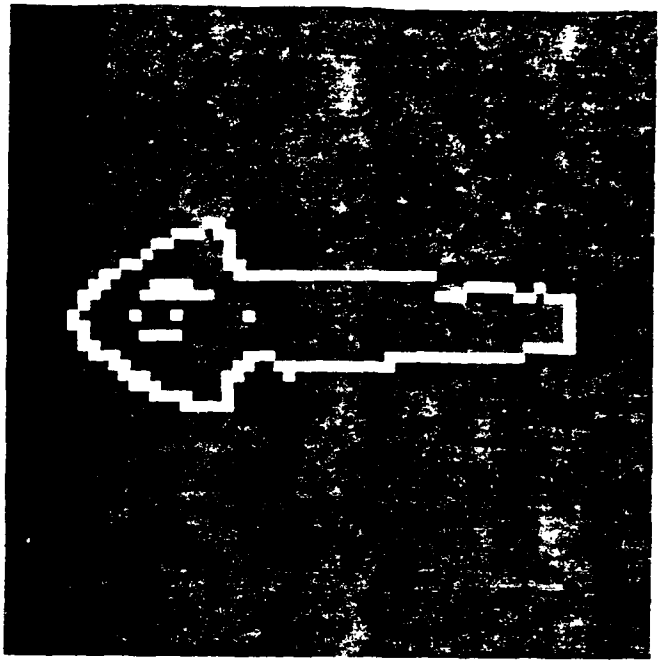


velocity diagram

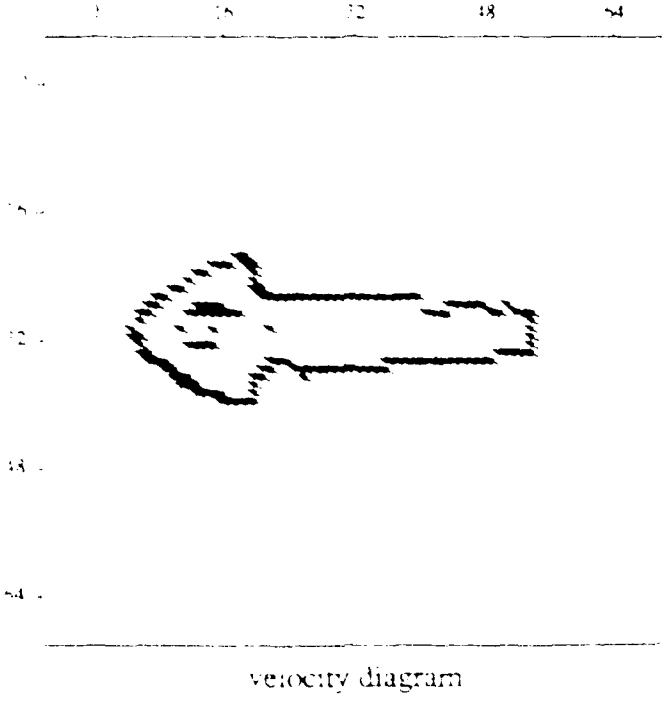
Figure II.6 Simulation of motion detection algorithm: synthetic image



original image



edge image



velocity diagram

Figure II-7 Simulation of motion detection algorithm: real image

The computational complexity of the implementation of the Motion/Edge Detection algorithm using the above described filtering techniques is now examined. As mentioned above, from Fig. II.3 it is apparent that two temporal filtering operations are necessary (t0 and t1). The required spatial filtering can be achieved by forming nine hierarchies, five associated with the t0 tree structure and four associated with the t1 tree structure. Additionally, 13 combinations are required (associated with all the x1's, x2's, x3's, y1's, y2's, and y3's shown in Fig. II.3.

The sum of the sizes of the filters used in each hierarchy is 41 (7 + 7 + 7 + 5 + 5 + 5 + 5). Each combination requires two operations and with four channels the computations associated with the combinations are scaled by $8\,N^2$. The resulting computational complexity is given by:

$$\begin{aligned} &9 \text{ Hierarchies} \cdot 41 \cdot N^2 = 369\,N^2 \\ &13 \text{ Combinations} \cdot 8 \cdot N^2 = 104\,N^2 \\ &2 \text{ Temporal} \cdot 3 \cdot N^2 = 6\,N^2 \end{aligned}$$

This totals to $479\,N^2$ multiplications and additions required to perform the filtering required by the algorithm with four channels. When subsampling is also used this amount can be reduced to approximately $250\,N^2$. The complexity calculated earlier in which the filter was implemented using conventional techniques was found to be $1602\,N^2$. Thus, utilization of the three filtering techniques described above results in a reduction in computational complexity of over 84 percent.

5. SIMULATIONS

Extensive simulations have been performed with the motion/edge detection algorithms using both synthetic and real images. In the reports *Software tests-Description* and *Software test-Simulations*, simulations are discussed in great detail.

Figures II.6 and II.7 show the results of simulations with synthetic and real images, respectively. It can be seen that these simulations produce good results. The value of σ plays an important role in obtaining good results and that is why it is extremely important to have a set of values of σ that corresponds to the range of velocities that can be expected in our application.

SECTION III

IMAGE SEGMENTATION

1. INTRODUCTION

This section contains a brief review of the segmentation subsystem and the algorithm created to perform the segmentation.

The input to this subsystem is the output of the motion detection subsystem, i.e. a binary edge image with a velocity associated with each output edge pixel. Its output consists in a set of segmented sub-images about each object in the input image. Ideally, there is just one object contained in each sub-image (or "window").

2. SEGMENTATION ALGORITHM

The basic method used by the segmentation algorithm can be summarized by the following five steps :

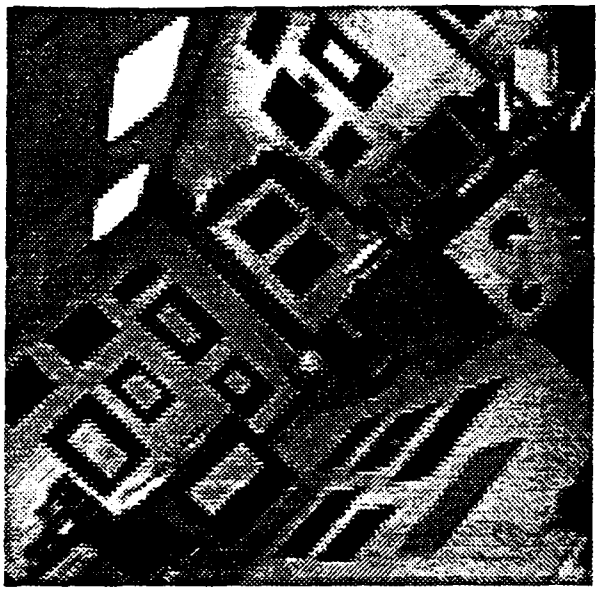
1. Cycle through pixels across the rows and down the columns.
2. Classify each pixel as belonging to an object or not.
3. Assume "adjacent" pixels are part of the same object if :
   a. Both pixels are edge pixels.
   b. Both pixels have associated with them velocities which are similar.
4. Merge different objects as necessary.
5. Update object boundaries as necessary.

In step three, the meaning of "adjacent" is flexible. Adjacent pixels could be taken to be pixels which are immediate neighbors of each other, or it could allow for single or multiple pixel gaps. The particular meaning of adjacent is set by a parameter in the segmentation algorithm.

Once all the pixels have been examined and classified appropriately, indices which define a rectangular window will have been found for each object present in the input image.

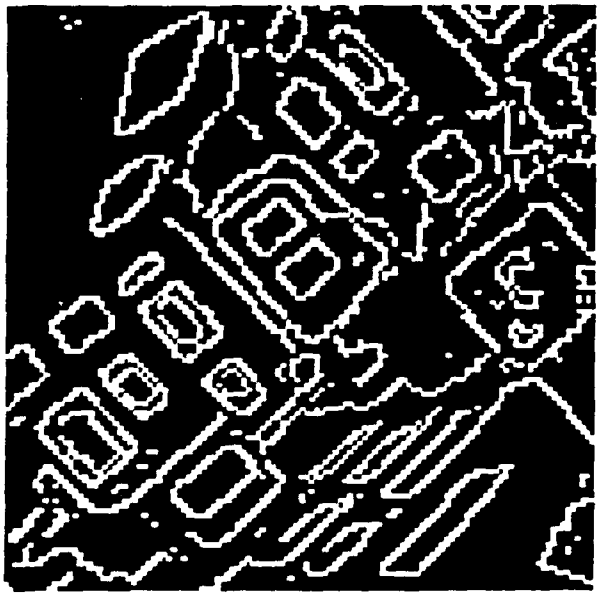
a. Simulations

Figures III.1a and b show frames one and three of an image sequence obtained with a camera mounted on the end effector of a robotic arm. The arm was moving and the end effector was rotating, a very involved situation. Figure III.1c shows the edge image obtained from the motion/edge detection algorithm, and Fig. III.1.d the velocity diagram from the same algorithm. Figure III.2 shows objects segmented by the segmentation algorithm using the information from Figs. IIIc and d. It can be observed that the results are very satisfactory even in the complicated case. Rotating objects present potential problems because the velocities at different pixels is not the same, see Sect.3.a below.

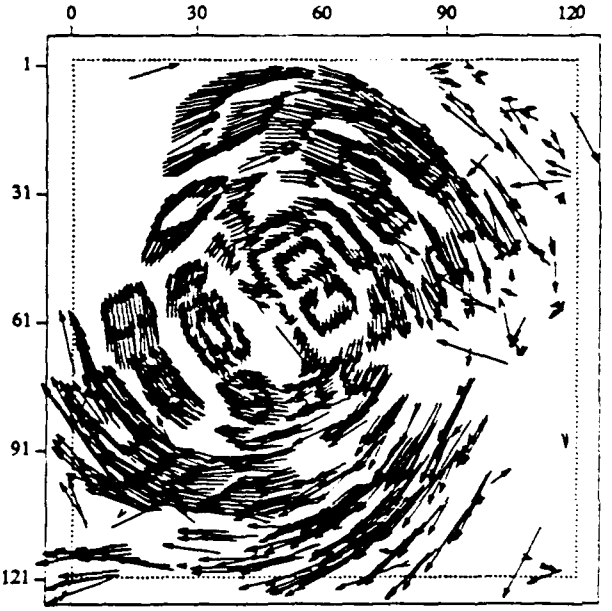


Original Image : Frame 1

Original Image : Frame 3

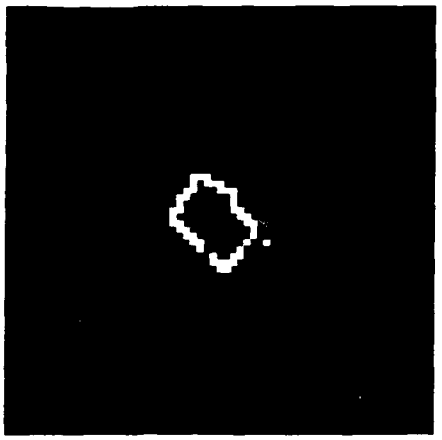


Edge Image

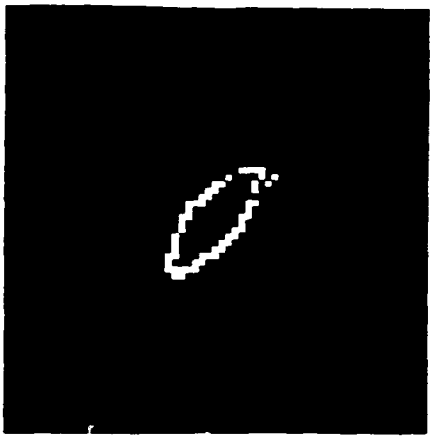


Velocity Diagram

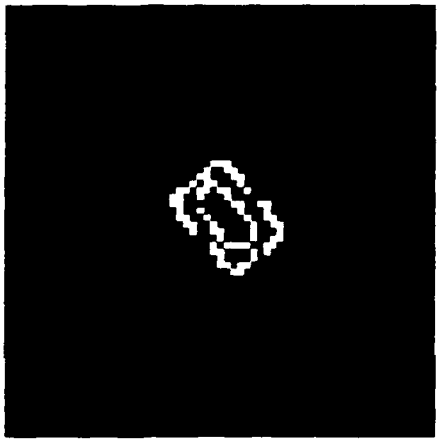
Figure III.1 Results for robot sequence, image size : 121x121



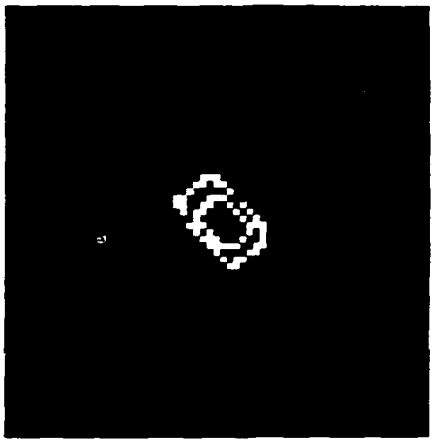
subimage 1



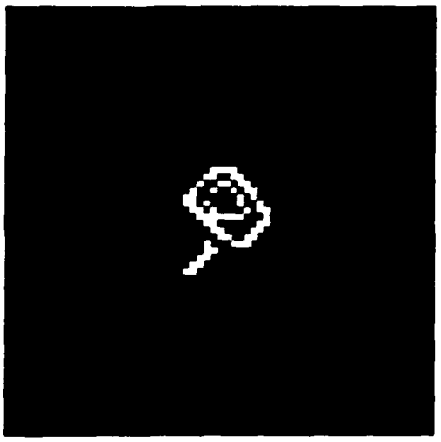
subimage 2



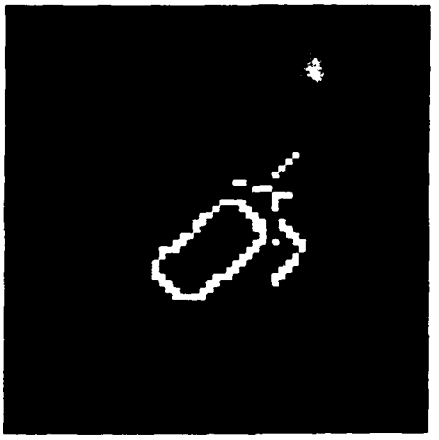
subimage 3



subimage 4



subimage 5



subimage 6

Figure III.2 Segmented subimages from the images of Fig.III.1

3. PROBLEMS

With the basic understanding of the segmentation algorithm described in Section 2 in mind, two problems will now be addressed. The first deals with rotating objects for which different pixels have different velocities. The second refers to windows that contain a main object and parts of other objects.

a. Segmentation Problems

First the situation of rotating objects will be examined. Figure III.3 illustrates this situation. Shown is an image of a rectangle rotating counterclockwise about its center. The numbers within the grid refer to the magnitude of the velocity at that location. The arrows indicate the direction of the velocity. As can be observed all of the pixels which make up the rectangle have associated

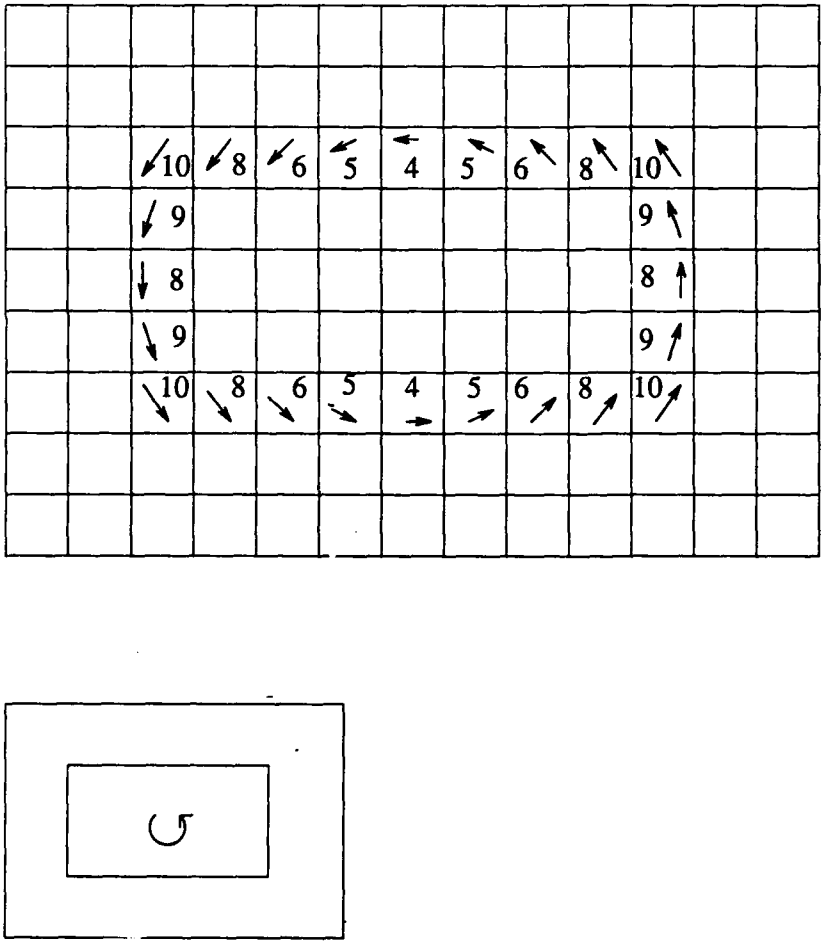


Figure III.3 A rotating rectangle

with them velocities which are different. For example, pixels located on opposite sides of the rectangle have velocities which are in directly opposite directions. However, pixels which are near one another have similar velocities. Since the segmentation algorithm only requires that adjacent pixels have similar velocities, rotating objects do not cause the algorithm any difficulties. Each pixel which makes up the rectangle illustrated in Fig. III.3 would be classified as being part of the same object.

The situation shown in Fig. III.3 is not unique, indeed any situation in which an object is rotating will share the same characteristic that adjacent pixels which make up the object will possess similar velocities. This will be true whether the object is rotating about its center or about any other point within the image. Thus, the algorithm as presented before did not have to be modified to handle rotating objects.

The algorithm was tested using synthetic data to mimic a rotating object and was found to work successfully, as expected. Later in this report results will be shown for situations in which a sequence of images containing a rotating object is input to the Spatio-Temporal Filtering subsystem and the output of this subsystem is subsequently fed to the segmentation subsystem.

The second question concerned situations in which nearby objects are present in an image such that rectangular windows which contain the objects overlap. Figure III.4 illustrates this situation. Shown is an image in which a square and a triangle are positioned such that rectangular windows which contain each of these two objects overlap. The large diagram in figure four represents the output of the segmentation algorithm once all the pixels have been examined. Pixels which have been found to be part of an object are given a value equal to the object number. Thus, all of the pixels marked with a "1" have been classified as belonging to object "1", and all of the pixels marked with a "2" have been classified as belonging to object "2".

Previously, the output of the segmentation algorithm consisted only of the row and column indices which defined rectangular windows which contained each object found in the input image. However, since it is desirable to only have one object in each segmentation window, and since simply extracting a rectangular window of pixels about an object can sometimes lead to situations in which more than one object is present in the window, the output of the segmentation algorithm had to be modified. The algorithm was modified as follows: an output edge image is formed for each object found by creating an edge image the same size as the rectangular window which just contains the object, however, only those pixels within the rectangular window which have a value equal to the object number are deemed to be edge pixels. All other pixels will be assigned a value of zero. The edge pixels are assigned a value of 255. By forming the edge image in this fashion one is assured of only having one object in each segmented edge image.

The algorithm was tested using synthetic data to mimic situations in which objects are nearby and was found to work as desired.

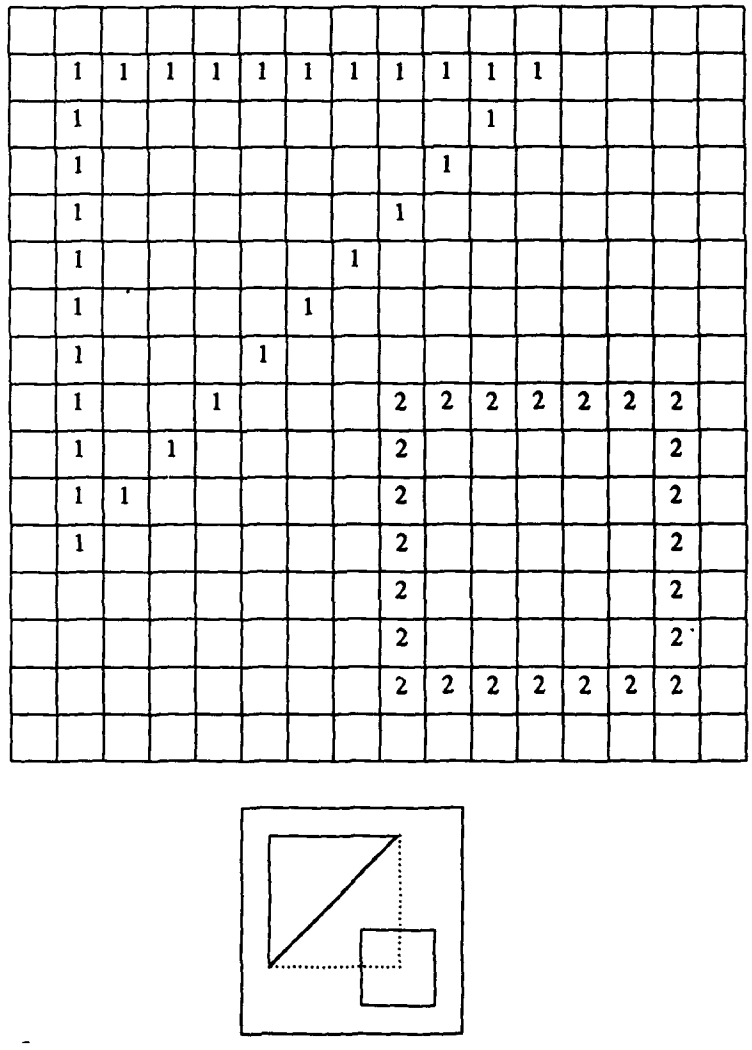


Figure III.4 Objects in two overlapping windows

SECTION IV

CENTROID DETERMINATION

1. INTRODUCTION

The binary edge image output of the motion/edge detection block is the input to the segmentation algorithm which outputs windows containing the binary edge image of each one of the objects present in the image. The centroid-determinator obtains the centroid of the objects in each window, with respect to window coordinates and, by extension, to image coordinates. Since centroid location (with respect to the object) is invariant to object size and to rotation on the image plain, continued centroid calculation provides a reliable way of tracking the target.

2. CENTROID CALCULATION

In the physical world, the centroid of an object is the center of mass of the object. For the context of image processing, the image can be thought of as a thin, two-dimensional object, where the mass of a point is proportional to the image intensity at the point. Since the centroid is constant for any object according to some object frame of reference, finding the centroid of the object allows elimination of any translation. As a result, the object can be normalized with respect to translation. Calculating the log-spiral mapping (LSM) around the centroid then provides a method of reducing scalings and rotations to cyclic shifts, which can be handled by the recognition network.

An additional advantage of the centroid is its use for target tracking. Since the centroid location is related to the target location, keeping track of the centroid location for several images will provide information on the target's velocity (both speed and direction).

a.Centroid Overview

The centroid location, (η_i,η_j) of an object in an (i,j) coordinate system is calculated as:

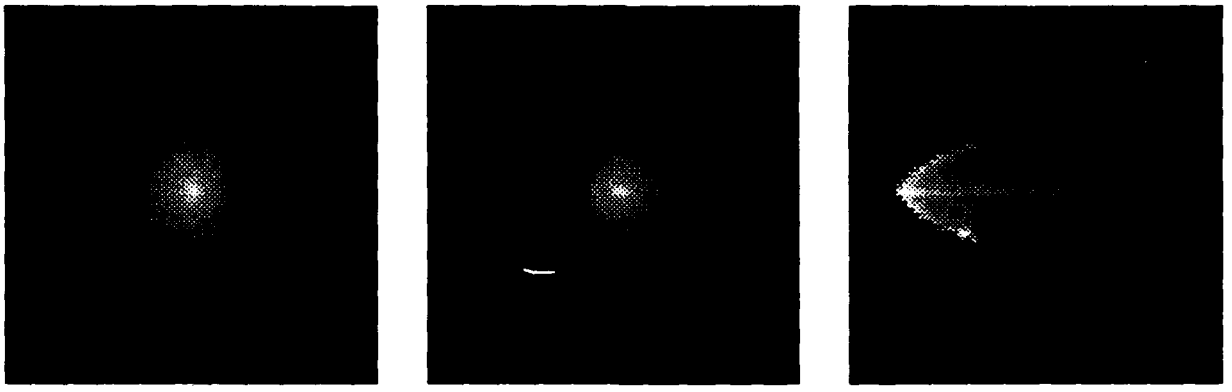
$$\eta_i = \frac{1}{M_T}\sum_{i=1}^{N}\sum_{j=1}^{N} m_{ij}\, i \qquad \text{(IV.1)}$$

$$\eta_j = \frac{1}{M_T}\sum_{i=1}^{N}\sum_{j=1}^{N} m_{ij}\, j$$

where M_T = total intensity mass = $\sum_{i=1}^{N}\sum_{j=1}^{N} m_{ij}$

m_{ij} = intensity of image pixel (i,j)

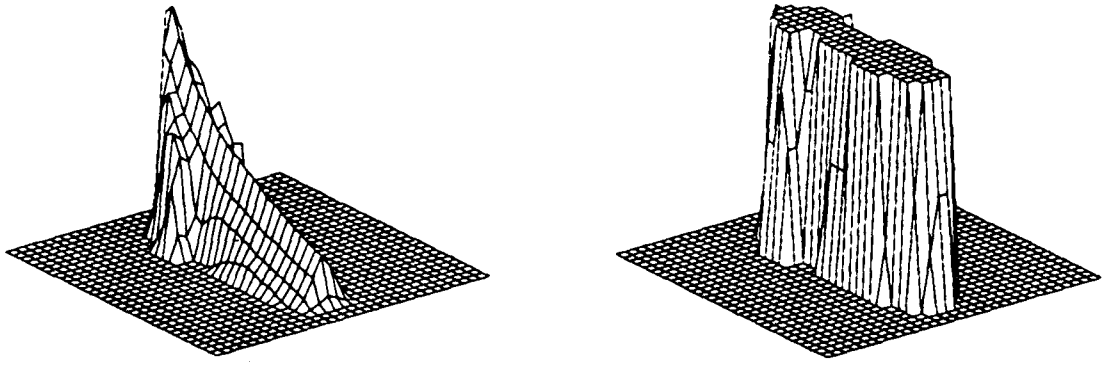
The image intensities of Eq.IV.1 correspond to the gray level image intensities of the missile plumes in Fig. IV.1 for the target seeker case. Although edge images are necessary for the LSM, they do not provide adequate information for centroid calculation based upon image area. As a result, the area intensities must be used to calculate the centroid locations.



a. 0 Degree Missile Plume b. 15 Degree Missile Plume c. 90 Degree Missile Plume

Figure IV.1. Missile Plume Images

For the ICBM target seeker application, gray level images provide better centroid results than thresholded binary images. The centroid of the gray level image of Fig. IV.2a is closer to the front of the missile plume (left side of figure) than the centroid of the binary image of Fig. IV.2b. Since the target seeker goal is a collision very near the hottest image point, a centroid weighted by image intensity will locate the centroid closer to the desired target point than a binary valued image. It is also hypothesized that the coolest plume gasses will oscillate in temperature near the binary threshold value, and that the resulting binary centroid will change frequently as a function of time, but inadequate data is available to test this hypothesis.



a. Gray Level Plume Image b. Binary Plume Image

Figure IV.2. Centroids for Gray Level and Binary Images (90 Degree Plume)

In order to calculate the centroid coordinates, in terms of its window (i,j) coordinates or in terms of its image (i,j) coordinates, an Analog Parallel Network (APN) had to be designed. The APN performs a spatial filtering to generate a peak value corresponding to the centroid location. A Hopfield network then determines the peak location. The centroid calculation approach is shown in Fig. IV.3. The network shown would find only the column location of the centroid. Finding the row location would require an equivalent network of summers, spatial filter, and Hopfield network.

The initial row of summing amplifiers in Fig. IV.3 is used to sum the pixel values in each column. The centroid of the resulting sum corresponds to the column location of the two-dimensional centroid. An equivalent process is performed to calculate the row location of the two-dimensional centroid. Separation of the two-dimensional problem into two one-dimensional problems is allowed due to the axes being orthogonal.

If the maximum sum corresponded to the centroid location, then the Hopfield net could be used immediately after the summers to find the peak/centroid location. The centroid and peak do not necessarily occur at the same location, though, so the spatial filter is needed to create a result whose peak location corresponds to the centroid location. This result is then used as input to a Hopfield network which is designed to have a logical "1" at the output which corresponds to the peak input. The Hopfield result is then given to a demultiplexer (DEMUX) to determine the centroid location in binary form.

3. SPATIAL FILTERING FOR PEAK DETERMINATION

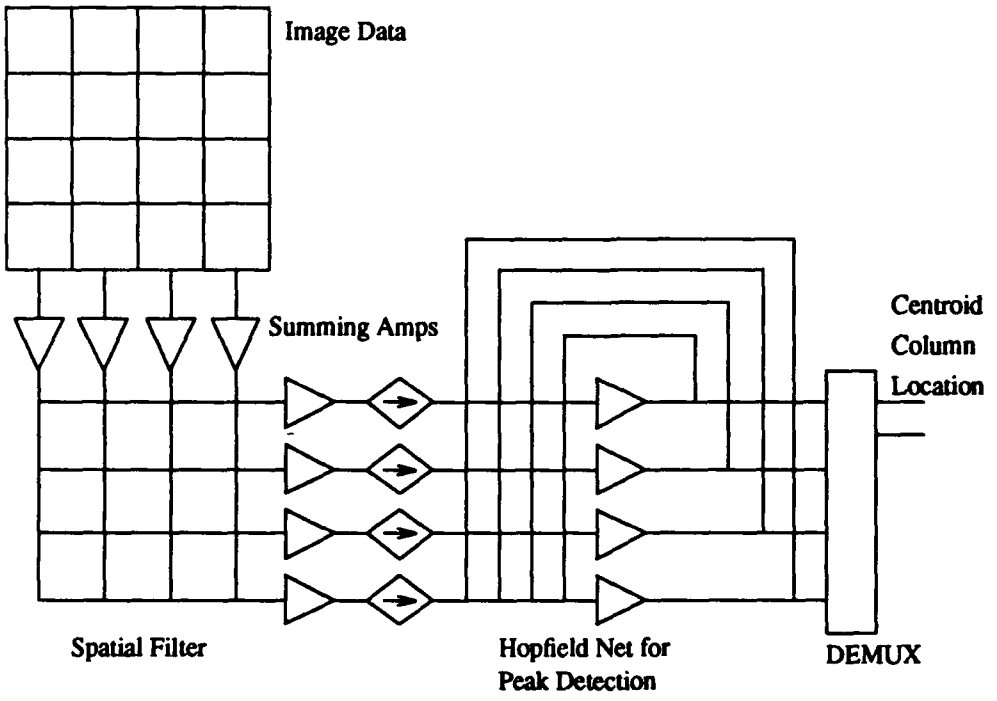


Figure IV.3. Network to Find Column Location of Centroid

Spatial filtering provides a novel approach to finding the centroid. A filter can be formed which, when convolved with the initial data set, will create a result whose peak occurs at the centroid of the data set. A useful property of convolution is that the centroid of a convolution result is equal to the sum of the centroids of the functions being convolved:

$$\text{Let} \begin{cases} \bar{x}_f = \text{centroid of f(x)}, & \text{where } f(x) = \text{data set} \\ \bar{x}_h = \text{centroid of h(x)}, & \text{where } h(x) = \text{filter set} \\ \bar{x}_z = \text{centroid of z(x)}, & \text{where } z(x) = f(x) * h(x) \end{cases}$$

$$\text{Then } \bar{x}_z = \bar{x}_f + \bar{x}_h \tag{IV.2}$$

If h(x) is symmetric about the origin, then this function will have its centroid at x = 0, and Eq.IV.2 will reduce to:

$$\bar{x}_z = \bar{x}_f, \quad \text{for } \bar{x}_h = 0 \tag{IV.3}$$

Consider a filter function, h(x), of the form:

$$h(x) = A - g(x^2) \tag{IV.4}$$

where A is some constant. The convolution result is given as:

$$z(x) = \int_{-\infty}^{\infty} f(\alpha)[A - g[(x-\alpha)^2]]d\alpha \tag{IV.5}$$

$$= \int_{-\infty}^{\infty} f(\alpha)Ad\alpha - \int_{-\infty}^{\infty} f(\alpha)g[(x-\alpha)^2]d\alpha$$

The first term corresponds to the area under the function, and is independent of translation. As a result, the peak value of z(x) occurs for the value of x which minimizes the second term. Therefore, the goal is to find a $g(x^2)$ which will minimize the second term. It can be shown that $g(x^2) = ax^2 + b$ will meet these requirements of minimizing the second term.

a.Spatial Filter Implementation

The spatial filter was implemented to determine the row and column centroids separately. The image pixel values are summed across the rows and down the columns as illustrated in Fig. IV.3. This reduction of the centroid calculation of a two-dimensional function into the calculation of the centroids of two one-dimensional functions greatly simplifies the complexity of the APN required to calculate the centroid.

Discrete space linear convolution consists of the following steps:

1. Reflecting the filter function, h(j), about the origin to create h(i-j) at i=0.
2. Finding the product of f(j)h(i-j) at each position j, where f(j) is the object density function.
3. Summing the products to obtain the convolution result for the i^{th} location.
4. Repeating the process for each location *i*.

Since the filter function derived is symmetric, step one does not change the appearance of the filter. For standard linear convolution, if f(i) is of extent N and h(i) is of extent M, then the convolution result is of extent M+N-1. Eq.IV.3, though, shows that the centroid of the result will equal the centroid of the object data set. Since the centroid of the object data set cannot be located outside of its own range, the examination of the convolution result is reduced to the extent of the object data set (N). The convolution is calculated as shown in Fig. IV.4.

The input voltages are supplied by the vertical lines and correspond to the data values, f(j). The connections are conductances which correspond to the filter values, h(i-j). Each row of connections corresponds to a single shift of the filter. The input voltages are multiplied by the conductance values, and the currents are added by the summing amplifier to obtain a current or voltage which represents the convolution result. An additional layer of inverting amplifiers is required if positive valued voltages are required.

4. HOPFIELD NETWORK FOR PEAK DETECTION

Although the convolution network filters the data set so the peak value corresponds to the centroid location, it does not explicitly indicate the location of the peak. Since the output is in analog form, it cannot be directly used by a digital computer. A network is required which can determine the peak of the many valued signals and indicate the peak location with binary values.

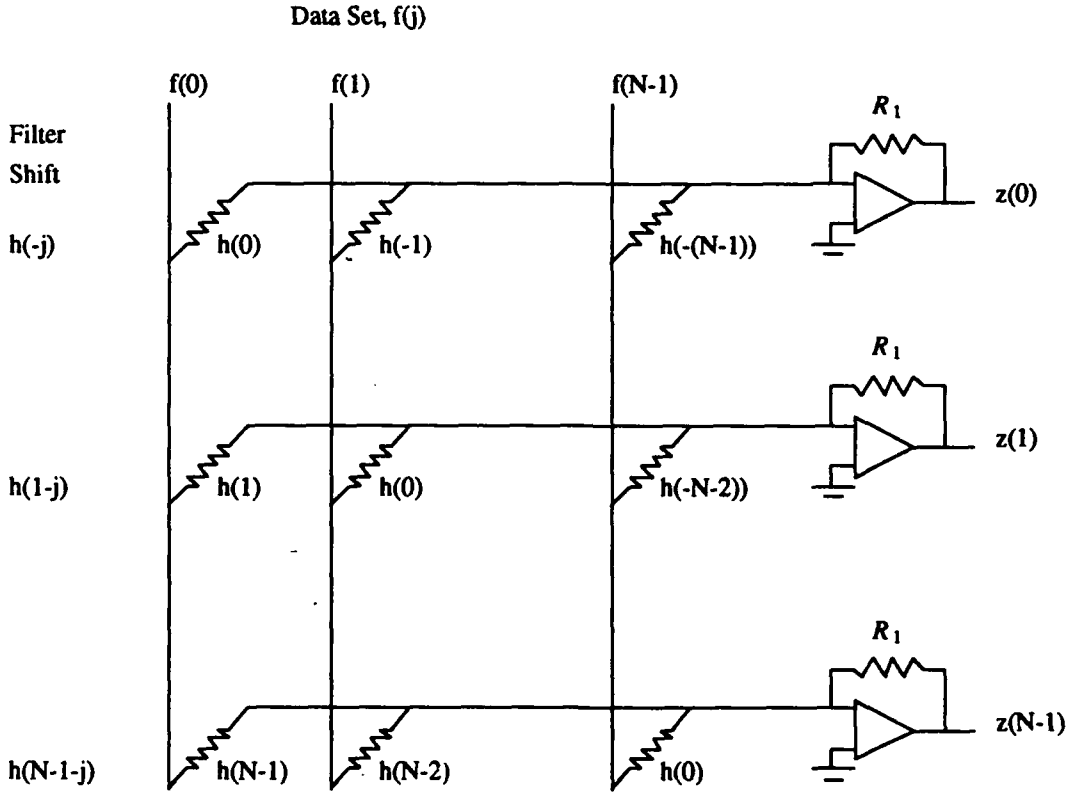


Figure IV.4. Parallel Analog Convolution Network

The Hopfield network was selected to implement this peak detection process.

Hopfield proposed a neural network capable of solving optimization problems [1,2,3,4]. The neural network layout is shown in Fig. IV.5. Note that the connections are shown in terms of their resistive values. In the following analysis, the connections will also be expressed in terms of their conductance values, T_{ij}, where $T_{ij} = 1 / R_{ij}$. Hopfield's amplifiers were characterized as having sigmoidal transfer functions. In addition, it was assumed that they had infinite input impedance.

The equations describing the network dynamics always lead to a convergence to stable states, and the states are the local minima of the quantity:

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} V_i V_j - \sum_{i=1}^{N} V_i I_i \qquad \text{(IV.6)}$$

$$= E_1 + E_2$$

The key to the use of a Hopfield network is to pose the problem to be solved in the form of Eq.IV.6, so that the function is minimized by the desired solution. Hence, the peak detection problem is solved by finding an assignment of input currents and conductances which will cause the network to converge at the desired solution. The desired peak detection solution is specified as follows:

1. The output is a logical 1 for the peak value.
2. Only one output is a logical 1. All others are logical 0's.
3. A single network should detect the peaks for many data sets.

Assigning the input currents as functions of the data set will meet the third criterion. If the currents are proportional to the data values, then the peak current will be some I_{peak} . If criterions 1 and 2 are met, then the second term of the energy function, will be minimized if the output voltage of the amplifier with the peak input current has a value of $V_{peak} = 1$.

The first term of the energy function must be used to guarantee criterion 2. Imagine the term was of the form:

$$E_1 = \frac{1}{2} B \sum_{i=1}^{N} \sum_{j=1}^{N} V_i V_j \qquad \text{(IV.7)}$$

$$\text{where } B = \begin{cases} c \text{ (some constant)}, & i \neq j \\ 0, & i = j \end{cases}$$

E_1 has a minimum value of 0 if all V_i terms are 0 or if only one V_i term is 1. Comparing this to the original energy function, it is seen that $T_{ij} = -B$, or

$$T_{ij} = \begin{cases} -c \text{ (some constant)}, & i \neq j \\ 0, & i = j \end{cases} \qquad \text{(IV.8)}$$

Although a negative conductance is difficult to fabricate, the equivalent effect can be obtained by using a positive conductance and connecting it to the inverted output of an amplifier. As a result, all of the Hopfield amplifiers are implemented with inverted outputs. Since a local minimum for E_1 does occur when all of the terms are 0, the input currents must be chosen

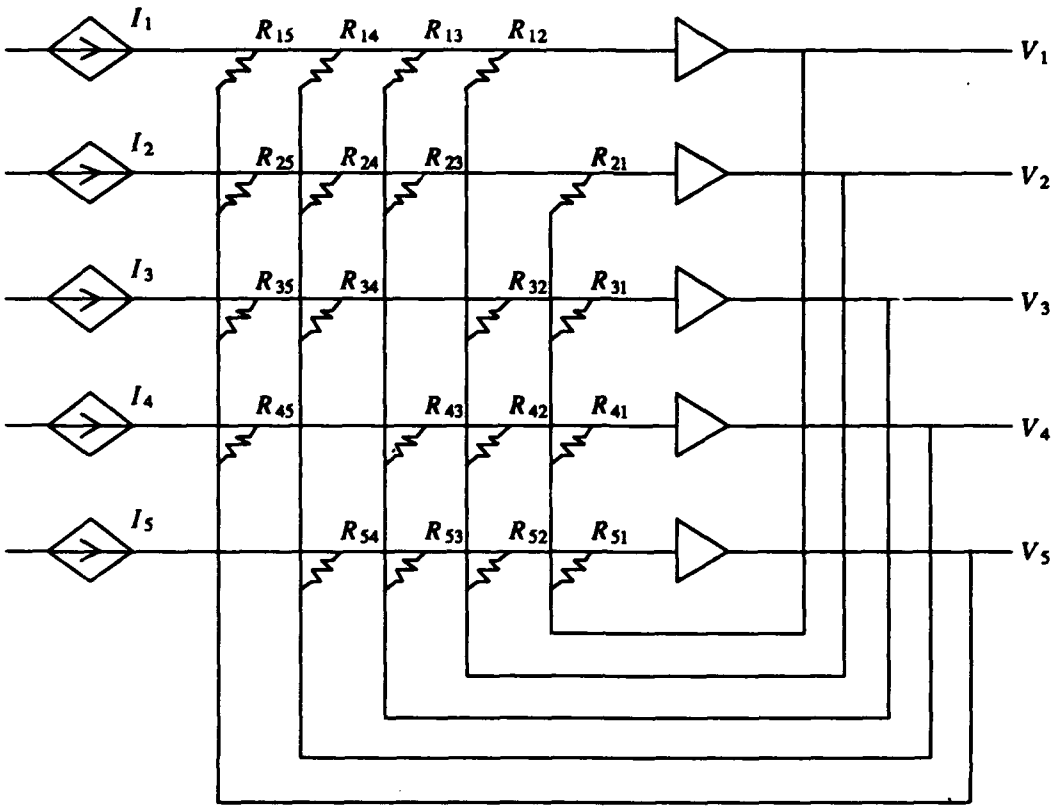


Figure IV.5. Hopfield Network

sufficiently large so the E_2 term prevents the total energy function from converging to the local minimum instead of the global minimum.

5.SIMULATION RESULTS

The centroid network was tested using the three missile plume images shown in Fig. IV.1. The results, listed in Table IV.1, demonstrate that the centroid network calculates the correct object centroids. Convergence times for the Hopfield network were less than 100 μs. SPICE simulations have also validated the program simulations for small image sizes and provide a degree of confidence that a hardware implementation of the network would indeed provide the correct centroid location.

As a result, centroid calculation provides a means of eliminating translation problems for the log-spiral mapping. Filtering a segmented window can create a result whose peak value occurs at the centroid of the input data set. This filtering has been shown theoretically and with simulations to provide the correct centroid location consistently.

Table IV.1. Centroid Results for 64 x 64 Missile Plumes

Image	Spatial Filter/Hopfield Results		Theoretical Location	
	Row	Column	Row	Column
0 Degree Plume	31	32	30.78	31.65
15 Degree Plume	31	35	30.78	34.78
90 Degree Plume	31	25	30.77	25.02

SECTION V

LOGARITHMIC SPIRAL IMAGING

1. INTRODUCTION

The log-spiral image is formed by combining and averaging rectangular image pixels. The log-spiral image is formed about a certain point in the rectangular image called the centroid (since this point is ideally the centroid of an object of interest within the rectangular image.) The log-spiral image is passed on to the multi-pixel recognition part of the vision system, since a log-spiral representation of an image facilitates the recognition of scaled and/or rotated objects.

The geometry describing a log-spiral tessellation of a digital image is more complex than that for a rectangular image. A review of the geometry of log-spiral images will be given and a constraint on the geometry will be derived which forms log-spiral pixels which have an aspect ratio of nearly one (pixels with an approximately square shape.) This is a desirable characteristic of digital images, since it yields an image with essentially uniform directional resolution. An example of a log-spiral image of a simple object is also shown.

2. LOG-SPIRAL IMAGE DESCRIPTION

A log-spiral image (LSI) is an image whose pixel boundaries are determined by exponentially spaced rings and equally spaced angular lines (radii) emanating from the center of the representation. The principal advantage of a log-spiral image over a rectangular image is that a log-spiral image, when mapped to a *computation plane*, results in invariance to scalings and rotation due to its polar organization of the visual information.

Figure V.1illustrates a log-spiral image tessellation. Four parameters determine the specific geometry of any log-spiral image. These parameters are:

R_{min} : The radius of the inner ring.

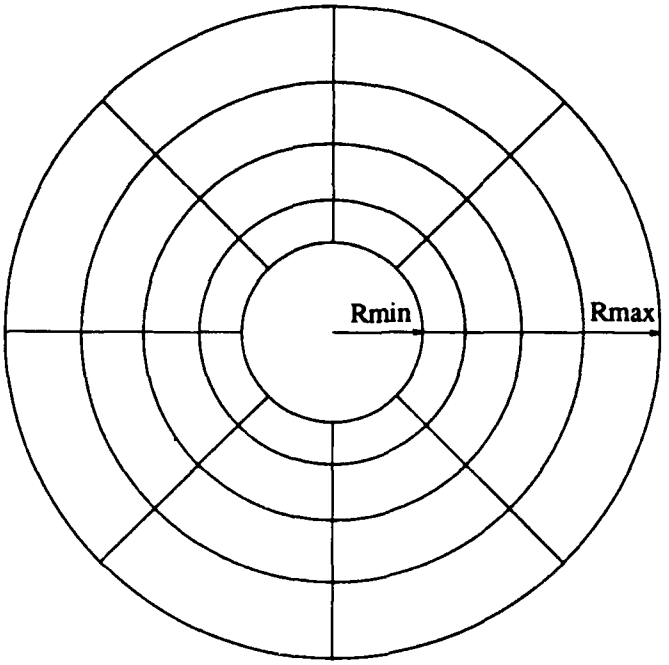
R_{max} : The radius of the outer ring.

ppr : The number of pixels per ring.

rng : The number of rings.

In Fig.V.1, the number of pixels per ring is 8, and the number of rings is 4. One unfavorable characteristic of a log-spiral image is that a "blind" spot exists in the center of the representation. The effects of this "blind" spot can be minimized by choosing R_{min} to be small compared to R_{max}. Selecting R_{min} to be 1/10 of R_{max} will result in a "blind" spot which covers just 1% of the entire image. In many practical instances this effect will not be detrimental.

The equation which determines the radii of each ring in an LSI image based on the parameters mentioned above is given by:



rng = 4, number of rings
ppr = 8, number of pixels per ring

Figure V.1. Log-spiral image tessellation.

$$R(i) = R_{\min} \cdot (R_{\max} / R_{\min})^{i/rng} \tag{V.1}$$

It is clear from Eqn V.1 that the rings are exponentially spaced and also:

$$R(0) = R_{\min} \tag{V.2}$$

$$R(rng) = R_{\max} \tag{V.3}$$

When a log-spiral image is being formed by combining and averaging rectangular pixels, $R_{\max}$ is typically chosen to be 1/2 the number of rows or columns (assuming a square array for the rectangular image) in the rectangular image. This gives values of radii in terms of pixel lengths in the rectangular image. $R_{\min}$ is typically chosen to be about 1/10 of $R_{\max}$. Choosing $R_{\min}$ to be smaller will reduce the size of the "blind" spot, but will also result in a larger exponential spacing (since $R_{\max} / R_{\min}$ will become larger. This can be undesirable since it leads to an LSI image with less total number of rings.

In general the parameters ppr and rng, referring to the number of pixels per ring and the number of rings, respectively, are free to be chosen. However, in most cases limits on the minimum size of the log-spiral pixels in the inner ring will restrict the selection of these

parameters.

A log-spiral tessellation results in pixels which all have the same aspect ratio. Since it is favorable to form digital images with pixels which are nearly square (directional resolution is more uniform in this case), a constraint equation can be developed relating ppr to rng, such that the aspect ratio of the log-spiral pixels is nearly one.

The aspect ratio for an arbitrary log-spiral pixel consists of a ratio between the radial length, L, of a pixel and the arclength, S, of the pixel. Thus, the aspect ratio is given by:

$$Aspect\ \ Ratio\ =\ \frac{L}{S} \tag{V.4}$$

where:

$$L\ =\ R(i)-R(i-1) \tag{V.5}$$

$$S\ =\ \frac{2\cdot\pi}{ppr}\cdot(R(i)+R(i-1))/2 \tag{V.6}$$

The equation for S is gotten by using the equation for arclength given by $s = r\cdot\theta$, where r is equal to the average value of the inner and outer radii of the pixel and θ is given by the angular extent of each pixel: $2\cdot\pi/ppr$.

By setting the aspect ratio, L/S equal to one, and by using the above relations for L and S and the previous expression given for the radius of the i^{th} ring (Eqn. V.1) it is possible to derive a relationship between rng and ppr such that the log-spiral pixels will all have an aspect ratio nearly equal to one. The result is:

$$rng\ =\ \frac{\log\left(R_{\max}\ /\ R_{min}\right)}{\log\left[(1+ppr/\pi)\ /\ (ppr/\pi-1)\right]} \tag{V.7}$$

As an example, with $R_{\max}\ /\ R_{\min}\ =\ 10$ and ppr = 64, the required value of rng to assure log-spiral pixels with aspect ratio nearly one is : rng = 24.

3. LOG-SPIRAL MAPPING (LSM)

Once an image is represented in LSI form in the *image plane* , a conformal transformation called a **log-spiral mapping (LSM)** given by

$$w\ =\ ln(z) \tag{V.8}$$

where

$$z\ =\ x+jy \tag{V.9}$$

and

$$w = u+jv \tag{V.10}$$

is performed.

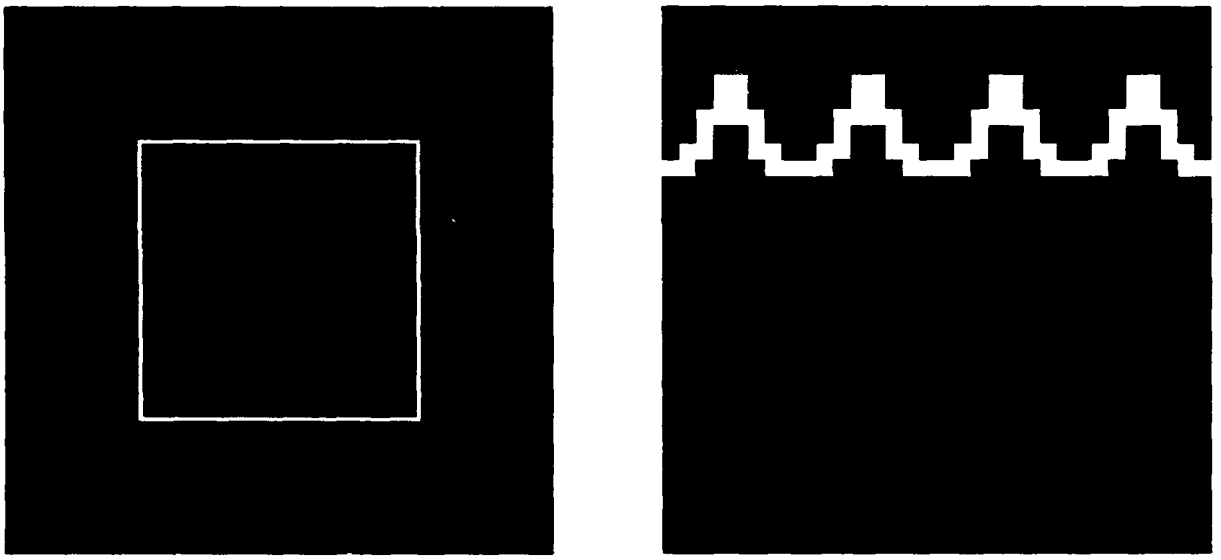
The LSM can also be expressed as

$$|w| = ln(r) = \ln[sqrtx^2+y^2]=u \tag{V.11}$$

and

$$\phi = arctan(y/x)=v \tag{V.12}$$

Figure V.2 shows a simple edge image of a centered square and its corresponding LSM image is shown in the computation plane. We have adopted the convention of magnitude along the vertical axis and phase along the horizontal.



(a) (b)

Figure V.2. Original image (a) and its log-spiral image (b).

As explained in the introduction, LSM images are used by the MHONN and the LCT. The rectangular to LSI transformation and the LSI image plane to computation plane conformal transformation can be performed either in software or with special digital circuit hardware. Another possibility for the former is the use of a VLSI LSI sensor.

3. A HARDWARE IMPLEMENTATION OF LOG-SPIRAL SENSOR

Using two separate CCD arrays to obtain log-spiral and rectangular information has its drawbacks -- most notably, size and cost. To have a single CCD camera provide information in both rectangular *and* polar formats, it would be necessary to use the approach of summing individual rectangular elements to form larger polar pixels. The following is a way of achieving this.

Directly "behind" the original set of shift registers would lie a second set of registers. The primary set would be interconnected so that, when the data is shifted out, it is delivered row-by-row to provide the data in a rectangular format. The second set of registers would have an interconnection scheme which reflects the polar pixel layout. That is, the picture elements would be shifted out of the CCD array in groups which form the polar pixels. When the polar data is shifted out (simultaneously with the rectangular data), all that must be known is the order in which the polar pixels are being extracted and the number of rectangular elements which constitute each polar pixel. Because the polar pixels increase in size exponentially as their distance from the center of the sensor increases, so will the number of individual pixels which must be averaged to calculate the intensity of that pixel.

Figures V.3 and V.4 show two possible averaging schemes for the extracted data. Both are essentially the same, with the only difference being that one uses an analog summing arrangement while the other uses digital hardware. The second approach appears to be more robust and so is advocated. Operation of the two is detailed. In both cases, the ROM holds one word of data for each polar pixel in the sensor. That data is the number of individual elements whose centers fall within the defining boundaries of the polar pixel. This information is termed the "run-length" for the pixel.

a. Operation of the analog summing arrangement (Fig V.3)

1) The count-down and S registers are loaded with the run-length of the next polar pixel being shifted out of the CCD.
2) As data is shifted out of the sensor array (and into the summing hardware), the count-down register is decremented. When the register reaches zero, the output of the Op-amp (which has been "summing" the pixel data) is fed to the A/D converter, the ALU divides that digitized sum by the run-length for that pixel (thereby performing the averaging), the OP-amp dump switch is activated (zeroing the output), and the next run-length is loaded.
3) Operation continues until all pixels in the current frame have been read.

b. Operation of the digital summing arrangement (Fig V.4)

1) At the beginning of a pixel-summing cycle, the A register is zeroed, and the count-down and S registers are loaded with the run-length of the next polar pixel.
2) As each rectangular element is shifted in, it is passed through the A/D converter and summed in ALU1. The results of each addition are placed in register A so that a cumulative sum may be obtained. Also, the count-down register is decremented.
3) When the count-down register indicates that the last element has been summed, the total is divided (in ALU2) by the run-length for that pixel (stored in the S register), thereby forming the average pixel intensity.

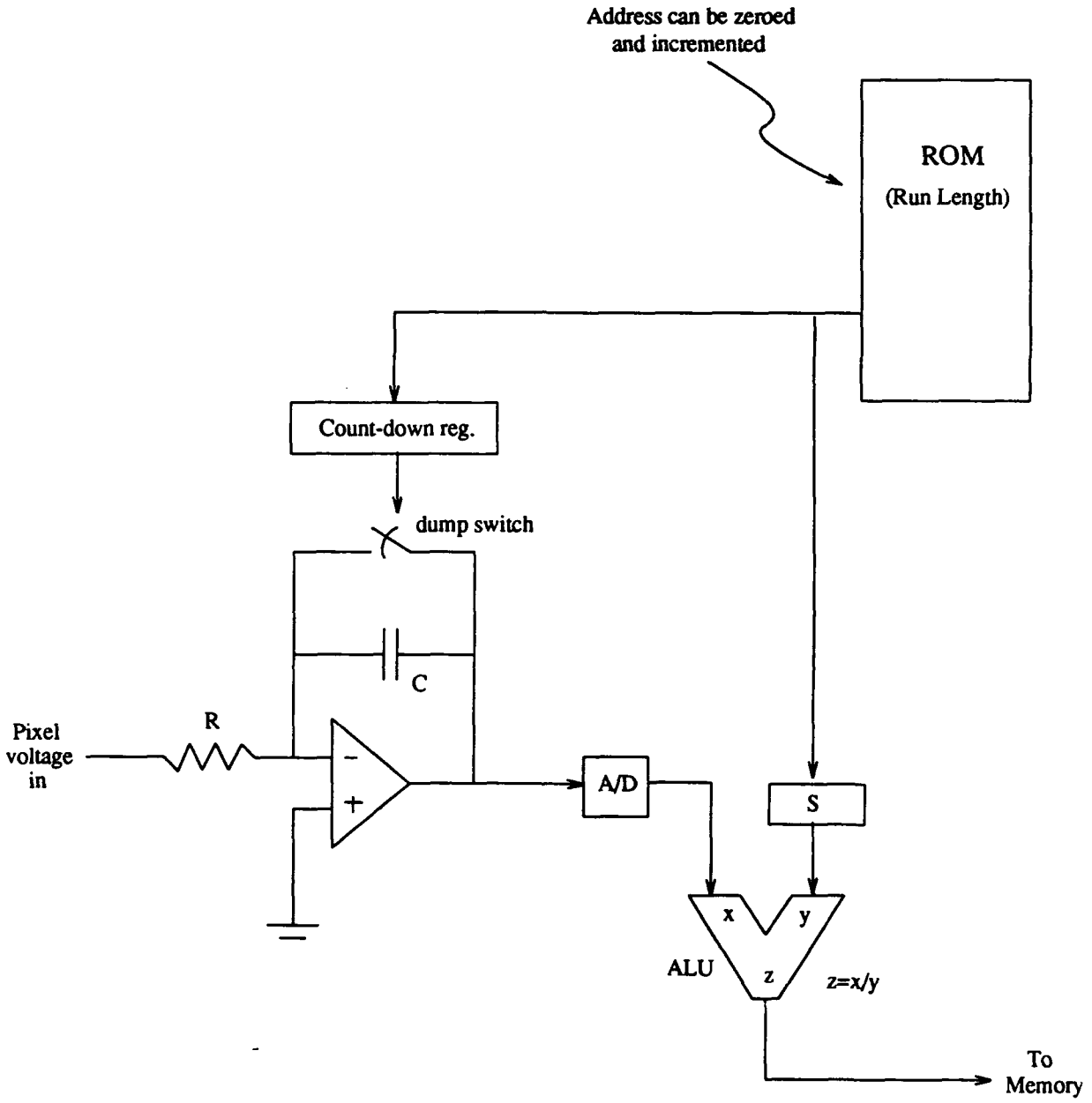


Figure V.3. Analog pixel-summing arrangement.

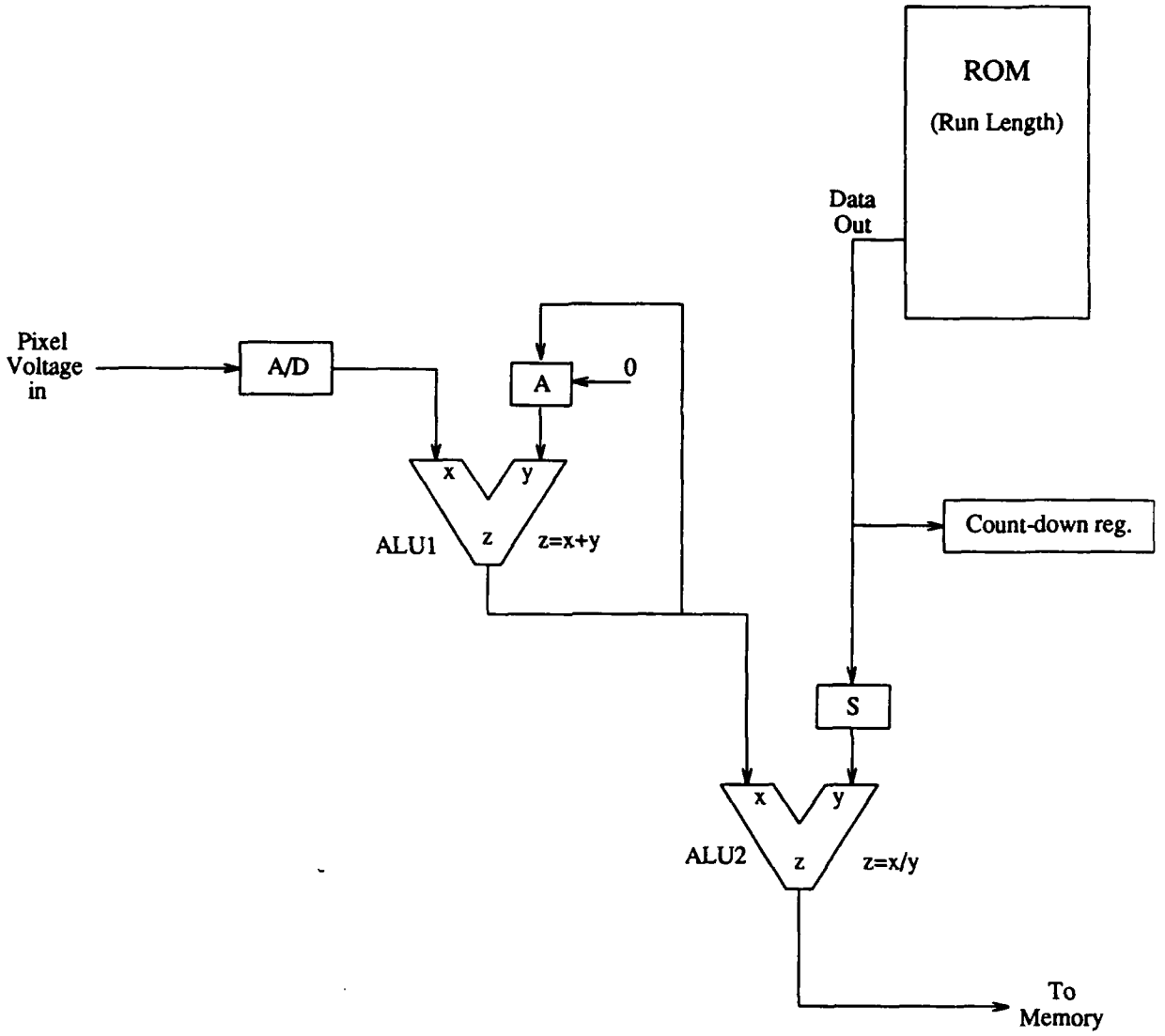


Figure V.4. Digital pixel-summing arrangement.

4) The cycle continues until the entire frame has been read.

A simple example of this operation follows. We will use the second approach (digital summing) and detail the operation in a clocked fashion. Assume a single- or multi-phase clocking arrangement in which, during each complete clock cycle, a single rectangular element is shifted out of the sensor and both ALUs are able to perform their required functions (if called upon to do so). We also assume that the hardware requires one extra clock cycle to perform the set-up operations necessary prior to summing the elements which form a particular pixel. For this example, the run-lengths of the first three polar pixels to be extracted (all that we will consider at

this time) are 4, 17, and 32 (completely arbitrary at this point since we do not know the layout of the polar pixels). Simulation is as follows (clock cycle followed by functions performed):

1) Zero ROM address. Load count-down and S registers with run-length of first polar pixel to be shifted out (4). Clear the A register.
2) Shift first rectangular element into A/D converter (flash converter) and add output to value in the A register (0). Store result in the A register. Decrement count-down register. Count-down register does not equal zero (yet) so continue.
3) Shift next rectangular element into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register.
4) Same as 3).
5) Same as 3). However, now the count-down register is zero. Therefore, divide the output of ALU1 by the contents of the S register (the run-length) in ALU2 to form the average pixel intensity. Store this in memory.
6) Increment the ROM address to point to the next run-length. Load the count-down and S registers with the next run-length (17). Clear the A register.
7) Shift first rectangular element of the next polar pixel into the A/D converter and add output to value in the A register (0). Store result in the A register. Decrement count-down register.
8) Shift next rectangular element into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register.

continue . . .

23) Shift next rectangular element (17^{th} in this pixel) into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register. Count-down register is now zero. Therefore, divide the output of ALU1 by the contents of the S register (17) in ALU2 to form the average pixel intensity. Store this in memory.
24) Increment the ROM address to point to the next run-length. Load the count-down and S registers with the next run-length (32). Clear the A register.
25) Shift first rectangular element of the next polar pixel into the A/D converter and add output to value in the A register (0). Store result in the A register. Decrement count-down register.
26) Shift next rectangular element into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register.

continue . . .

56) Shift next rectanguler element (32^{nd} in this pixel) into A/D converter and add output to value in the A register. Store result in the A register. Decrement count-down register. Count-down register is now zero. Therefore, divide the output of ALU1 by the contents of the S register (32) in ALU2 to form the average pixel intensity.

Store this in memory.

57) Increment the ROM address to point to the next run-length. Load the count-down and S registers with the next run-length. Clear the A register.

Continue until entire array is read.

This algorithm is very systematic and easily implemented, as shown above.

Since pixels on the innermost ring of the polar arrangement have the smallest area, they will be formed from the smallest number of rectangular elements. Consequently, all timing considerations and error analysis must be performed using these pixels. That this is so for error analysis is evident from an averaging point of view. Since the polar pixel's intensity will be the average intensity of the rectangular elements whose centers lie within its border, the smaller the area of the polar pixel, the greater will be the error (or error variance) of the estimated intensity. As for timing considerations, since one clock cycle is wasted in preparation for each polar pixel (preparing the hardware to sum and average the incoming rectangular elements), the smaller the polar pixel area, the greater will be the clock-cycle overhead expended. Hence, the dependence of timing considerations on the innermost ring of polar pixels (smallest polar pixels) is justified.

SECTION VI

MULTI-PIXEL TARGET RECOGNITION

1. INTRODUCTION

This section presents the design of a shift invariant pattern recognizer based on a modified higher-order neural network (MHONN). When the MHONN is integrated with the centroid calculation and the LSM subsystems, translation, rotation around the optical axis, and scaling invariant pattern recognition can be achieved by this integrated system. The design objective is to deal with large-scale images with possible pattern deformation, noise and highly textured background.

In this report we emphasize the important aspects of the MHONN algorithm development and simulation results. Hardware design can be found in *E.Subsystem Report: Analysis*.

2. POSITION IN THE INTEGRATED VISION SYSTEM

With reference to Fig. I.1, Section I of this report, we see that the MHONN subsystem uses as its input the LSM computation plane images (one windowed image per object.)

This section describes the MHONN design. The MHONN is the most vital part of the overall system. Its input is a LSM binary edge image, and its output is a classification signal which identifies the object and allows further processing of the corresponding window. It is required to achieve reliable and robust pattern recognition independent of the possible pattern scaling and rotation (translation invariance is achieved by the centroid subsystem,) noise, and deformations. Detailed theoretical development and design considerations about MHONN are addressed in the report *E.Subsystems Report - Algorithms* . Here we describe the most salient aspect of the MHONN theory, design and simulation results.

3. MHONN DESIGN

As stated before, the centroid calculation eliminates the object translation in the original image; the LSM converts the object rotation in the original image into a horizontal cyclic shift (CS) and the object scaling into a vertical ordinary shift (OS) in the log-spiral mapped image. Therefore, the task of the MHONN is to achieve pattern recognition independent of this two types of shifts.

Reid [1] proposed a HONN with only the second-order terms to achieve ordinary translational invariant pattern recognition. The original second-order neural network developed by Reid consists of only two layers of neurons. As shown in Fig. VI.1, the first layer is the multiplier-layer, and the second layer is the output layer, in which the neural transfer function is a weighted sum followed by a Θ function as given in Eqn VI.1. The Θ function is a step function. The adjustable weights are between these two layers. Reid used one layer of adjustable weights and directly implemented Giles' [2] translational invariance constraints by setting certain weight values to be the same. For instance, in Fig. VI.1, $w^i_{(2,2),(1,4)} = w^i_{(4,1),(3,3)}$, and $w^i_{(1,2),(1,3)} = w^i_{(3,1),(3,2)}$, etc. For clarity of notation, in the following text, we use a superscript "i" to indicate the i^{th} output element (summer), and the ordered pairs in the subscript denote the location of the two pixels. In this architecture, Reid only utilized these constraints to reduce the

independent weights number, therefore speeded up the training process, but the actual number of weights was not reduced by the the invariance constraints. The advantage of this setting is that the network is very simple. However, the disadvantage is the huge memory requirement for weight storage.

In our system, even though the MHONN has to handle not only the OS in the vertical direction, but also the CS in the horizontal direction, the nature of the problem is still of second order (i.e., it is necessary and sufficient for the classifier to utilize the correlation information between each pair of input signals to achieve the desired recognition). So, a second-order neural network is used as the pattern recognizer. However, we have to make several major modifications to the original second-order network to meet our special needs as well as to improve the network performance. This constitutes one of out major original contributions. Before further discussion, the term **feature** must be defined for clarity and convenience. In this section of the report, **feature** is defined as a **vector** connecting two pixels in the input plane. The length and orientation of this vector specifies a unique spatial separation between a pair of pixels.

a. HONN Architectural Modification

As mentioned in the previous section, the input of the HONN is binary, so AND gates instead of multipliers are used to extract the correlation information between each pair of the input pixels, because the AND operation is much more computationally efficient and easier to implement in hardware. The first modification consists in adding a layer of summers in-between the AND layer and the output layer. The significance of this modification is two-fold: first, it

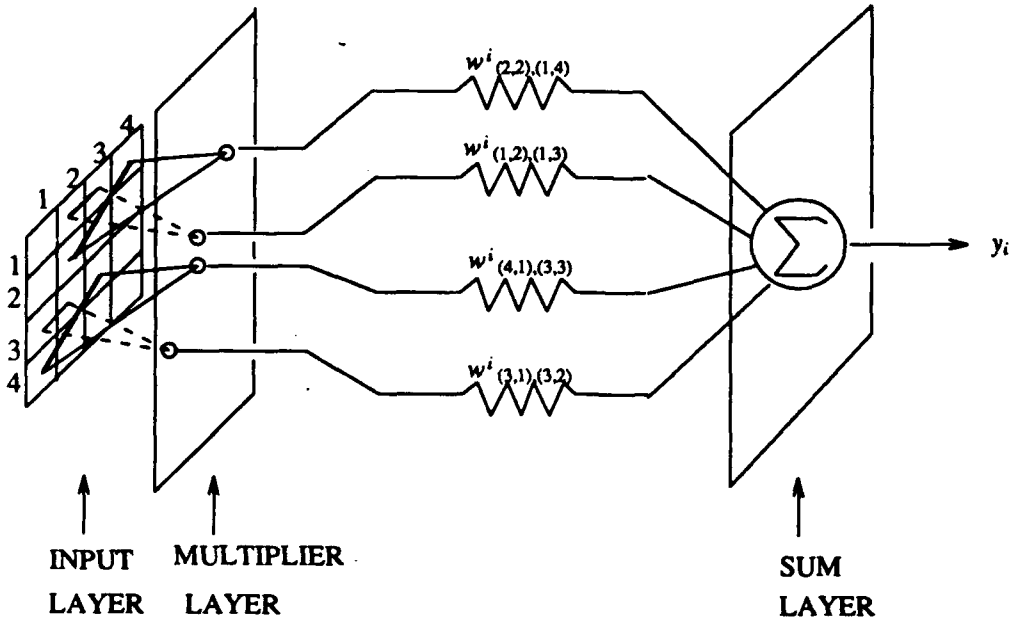


Figure VI.1 Original second-order neural network architecture.

reduces the number of weights in the by orders of magnitude, which means that the memory reduction in software implementation, and the number of components in hardware implementation is reduced in proportionally. Second, it reduces the computation time of the software simulated HONN.

Fig. VI.2 shows the architecture of the modified second-order neural network with translation invariance. In this architecture, the output signals of the AND gates corresponding to each unique type **feature** are first summed together, and then, the summed signal is fed into **one** adaptable weight.

From now on, the summers in the SUM_1 layer will be referred as "accumulators", and the output of this layer will be referred as "accumulator output pattern". The value of an accumulator output reflects the frequency of occurrence of a particular type of feature. Because the accumulators only record the occurrences of features but not their locations, the accumulator output pattern is translation invariant. From a pattern recognition viewpoint, the MHONN in Fig. VI.2 consists of two parts: the first part includes the AND_layer and the accumulator layer, the function of this part is to extract translation invariant features from the input image; the second part is composed of the adjustable weights and the output summers. This part is a one-layer neural network. Its function is to classify the translation invariant patterns generated by the accumulators.

The above seemingly simple modification can save a tremendous amount of memory space or hardware components. The analytical derivation is as follows: the general second-order neuron transfer function is given in Eqn.(VI.1):

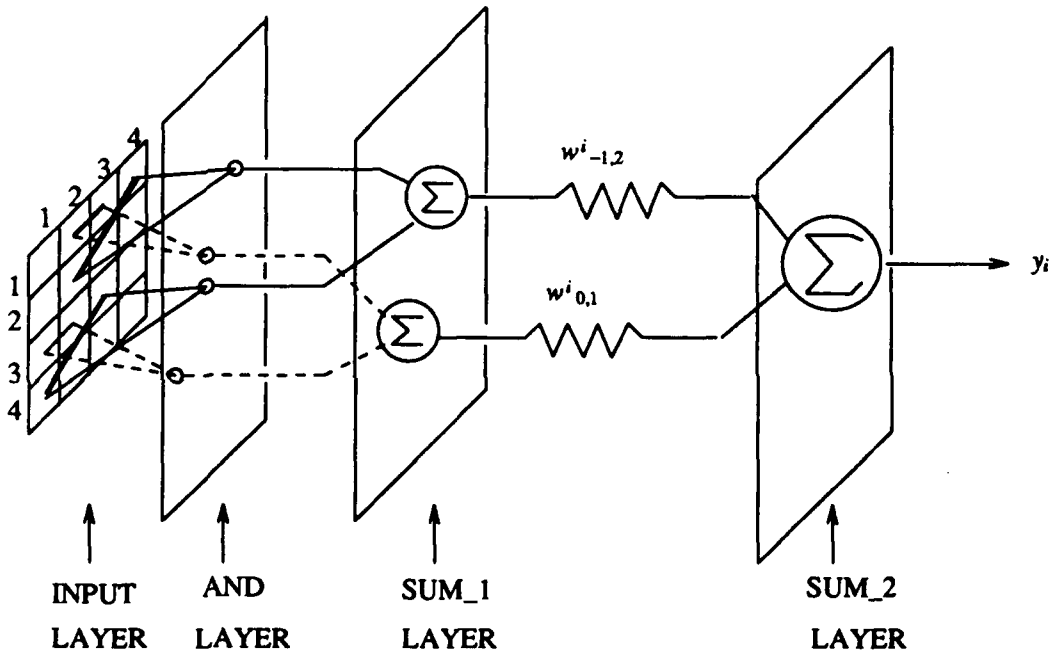


Figure VI.2 Architecture of the modified second-order neural network with translational invariance.

$$y_i = \Theta\left[\sum_{j=1}^{N}\sum_{k=j+1}^{N} w^i{}_{jk}\, x_j\, x_k\right] \qquad \text{(VI.1)}$$

where x_j stands for the j^{th} pixel located at row r_j and column c_j. After applying the translational invariant constraints (see Eqn. VI.9) onto the weight matrix $W_i = \left[w^i{}_{jk} \right]$, the weights $w^i{}_{jk}$'s will group into classes. Weights in each class will have the same value, no matter what the value might be. The above modification can be represented by rearranging the summations in the above equation according to the classes:

$$y_i = \Theta\left[\sum_{j=1}^{N}\sum_{k=j+1}^{N} w^i{}_{jk}\, x_j\, x_k\right] = \Theta\left[\sum_{mn} w^i{}_{mn} \sum_{v\,(m,n)} x_j\, x_k\right] \qquad \text{(VI.2)}$$

where $m = r_k - r_j$; $n = c_k - c_j$; (r_k, c_k) represents the row and column location of pixel x_k; $v(m,n)$ is a set of ordered pairs $(x_j\ ,\ x_k)$ in which the vector from x_j to x_k belongs to the same type of feature specified by $(\ m\ ,\ n\ )$. To be more precise, for each pair of (m,n), if $m = r_k - r_i$ and $n = c_k - c_j$, then $(\ x_j\ ,\ x_k\ )$ belongs to the set $v\ (m,n)$.

After this modification, the *perceptron learning rule* (which was used in Reid's HONN) can still be used to train the network. The weight values can be updated according to the following equation:

$$\Delta w^i{}_{mn} = \alpha\,(\ y_i{}^{d} - y_i{}^{a)} \cdot A_{mn} \qquad \text{(VI.3)}$$

where

$$A_{mn} = \sum_{v\,(m,n)} x_j\, x_k \qquad \text{(VI.4)}$$

where m , n , v are defined as before; $y_i{}^{d}$ is the desired i^{th} output element value, and $y_i{}^{a}$ is the actual i^{th} output element value. However, it will be shown in subsection VI.3.c. that a better alternative exists for the learning rule.

Obviously, if the input window size of the MHONN is N × N, the range of m and n are both from $-(N-1)$ to $(N-1)$. Therefore, the maximum possible number of independent weights is $(2N-1)^2$. However, m and n cannot be zero at the same time because we don't correlate a pixel with itself; in addition, according to Eqn (VI.1), once the pair $(x_j\ ,\ x_k\ )$ has been correlated, the pair $(x_k\ ,\ x_j\ )$ will not be correlated. So, the total number of independent weights is:

$$N_{indp.} = \frac{(2N-1)^2 - 1}{2} = 2\,(N^2 - N\,) = O\,(N^2\,) \qquad \text{(VI.5)}$$

For an N × N input image, the number of weights in the original HONN is $C_{N^2}{}^{2} = O\,(N^4\,)$, but after the modification, the number of weights in the MHONN is reduce to $2\,(N^2 - N\,)$, and the trade-off is only to add $2\,(N^2 - N\,)$ summers. This modification reduces not only the memory requirement by a factor of $O(N^2)$, but also the complexity of the simulated HONN. According to Eqn (VI.1), the complexity of the original simulated network $C_{org.}$ is:

$$\begin{aligned} C_{org.} &= C^2_{N^2}\ \textit{AND-operations} \\ &+ C^2_{N^2}\ \textit{summations} \\ &+ C^2_{N^2}\ \textit{multiplications} \end{aligned} \qquad \text{(VI.6)}$$

After the modification, the complexity $C_{mdf.}$ becomes:

$$\begin{aligned} C_{mdf.} &= C_{N^2}^2 \ AND\text{-}operations \\ &+ C_{N^2}^2 + 2(N^2 - N) \ summations \\ &+ 2(N^2 - N) \ multiplications \end{aligned} \tag{VI.7}$$

Assume that each AND-operation, summation, multiplication takes α, β, γ time units, respectively. When N is large, the complexity reduction factor C_r is:

$$\begin{aligned} C_r &= \frac{C_{N^2}^2 \cdot \alpha + [C_{N^2}^2 + 2(N^2 - N)] \cdot \beta + 2(N^2 - N) \cdot \gamma}{C_{N^2}^2 \cdot (\alpha + \beta + \gamma)} \\ &\approx \frac{C_{N^2}^2 \cdot (\alpha + \beta + \frac{4}{N^2} \cdot \gamma)}{C_{N^2}^2 \cdot (\alpha + \beta + \gamma)} \\ &= \frac{\alpha + \beta + \frac{4}{N^2} \cdot \gamma}{\alpha + \beta + \gamma} \\ &\approx \frac{\alpha + \beta}{\alpha + \beta + \gamma} = \frac{1}{8} \end{aligned} \tag{VI.8}$$

For a general purpose digital computer, $\alpha = 2$ (*clock cycle*), $\beta = 2$ (*clock cycle*), $\gamma = 28$ (*clock cycle*), so the complexity reduction factor is about 8.

b. Invariance Constraints Development

The ordinary translational invariant constraint developed by Giles [2] is stated here with new notation:

$$w^i{}_{jk} = w^i{}_{(r_k - r_j),(c_k - c_j)} = w^i{}_{mn} \tag{VI.9}$$

This constraint cannot handle the CS properly, because the CS is a special case of the ordinary translation, and it requires a stronger constraint.

As shown in Fig. VI.5, in the LSM computation plane, a feature F_1 in image (A) is cyclically shifted to the right by one pixel and results in feature F_2 in image (B). If CS invariant pattern recognition is desired, these two features should be regarded as the same feature, i.e., the weights connecting to these two features should be equal. However, if we apply the ordinary translational invariant constraint to images (A) and (B), we will have the following situation:

By scanning through the image column-by-column, from left to right, in image (A) we get:

$x_1 = pixel_a$, $x_2 = pixel_b$

and

$w_{F_1} = w_{(r_2 - r_1),(c_2 - c_1)} = w_{-2,3}$

in image (B) we get:

$x_1 = pixel_b$, $x_2 = pixel_a$

and

$w_{F_2} = w_{(r_2 - r_1),(c_2 - c_1)} = w_{2,1}$

Obviously, $w_{F_1} \neq w_{F_2}$, therefore, this constraint is not CS invariant.

As shown in Fig. V.38, in the LSM computation plane, a feature F_1 in image (A) is cyclically To develop the true CS invariant constraint, we must first have a close look at how the CS is different from the OS. As shown in Fig. VI.3, a CS in the horizontal direction (without losing generality, we can assume a right shift here) will not shift any signals on the image **out** of the image, but rather, wrap those signals back to the left side of the image. If we view the LSM computation plane image as a cylinder , i.e., let the left and right sides of the image meet, the CS becomes an ordinary translation on the cylindrical surface. An immediate observation is that on this cylindrical surface, each pair of pixels corresponds to two features: one feature is obtained by connecting pixels **a** and **b** counter-clock-wise(CCW), and other other one is obtained by connecting them clock-wise(CW). So the question is: which feature should we choose in order to achieve the CS invariance? The answer to this question depends on the convention we use to scan the image pixels.

- When the images are scanned column-by-column, from left to right:

In the ordinary translational invariant algorithm, the CCW feature (F_1) is **always** used to represent each pair of pixels. The pair (a, b) is represented by F_1, while after the cyclic right shift, the same pair is represented by another feature, F_3, because the pixel **b** is encountered before pixel **a**. Obviously, there two features are not equal, so we have $w_{F_1} \neq w_{F_3}$. Therefore, this way of choosing features cannot handle CS invariance properly. However, if we do not consider the directions (i.e, CCW or CW) of the connections and **only** choose the feature with the shorter length (or the longer length), the CS invariance is achieved.

Under the above image scanning convention, the invariant algorithm we used in the MHONN is the following: each feature is specified by a pair of (Δrow, Δcol), and each accumulator sums up the occurrence of one unique type of feature in the N × N input image. According to this algorithm, the ordinary translational invariant algorithm is first applied to each pair of pixels, which means the CCW feature is used; but if the Δcol of a CCW feature is greater than N/2, then CW feature is used; if the Δcol is equal to N/2, the CW feature and the CCW feature both have the same length, we pick the one with positive Δrow.

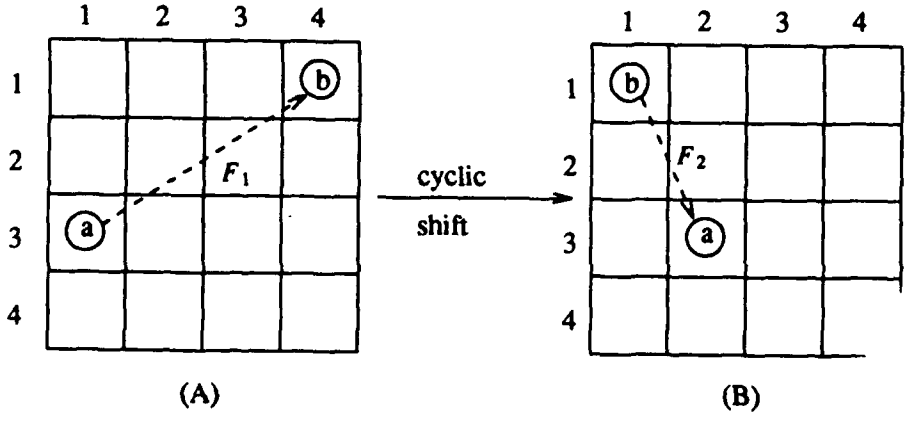


Figure VI.3 Cyclic shift

The CS, OS invariant weight constraint implementation algorithm discussed above is denoted as "MHONN Invariance Algorithm I". Algorithm I has been simulated with a program, and the simulation results show (see *Software Test Report -- Simulations* for data) that this algorithm can indeed handle the translation in the vertical direction and CS in the horizontal direction perfectly.

c. Learning Mechanism for the Modified HONN

In practice, pattern recognition requires the nonlinear separation of the pattern space into subspaces, with each class of patterns occupies one or more of the subspaces. It was found [3,4] that a first-order neural network with single layer of adjustable weights can only perform linear discrimination. However, either first-order network with multilayer adjustable weights (in between two consecutive layers of weights are the first-order hidden layer neurons, of course) or higher order network with single layer of adjustable weights can achieve the desired nonlinear separation in the pattern space. Therefore, one layer of adjustable weights is sufficient for the HONN or MHONN to perform nonlinear discrimination among the patterns.

In order to achieve the desired classification, the adjustable weights have to have "proper" values. The process which leads to the weights to have the proper values is called the learning process of the neural network. In other words, the objective of the learning process is to implement a desired mapping between a set of input patterns and a set of output patterns by adjusting the weights to the proper values. In Reid's original HONN with only one layer of adjustable weights, the *perceptron learning rule* was used. This learning rule is a special case of the *delta rule*. In the *delta rule* the input and the output of the network are not necessarily binary values, and in the output layer neurons, a sigmoid Θ function is used. In the perceptron learning rule, all the input and output are required to be binary values, and the output neuron use a step function instead of the sigmoid function. The *perceptron learning rules* for the second and the third order HONNs can be expressed as:

$$\Delta w^{i}_{jk} = \alpha \, (y_i^d - y_i^a) \, x_j \, x_k \qquad \text{(VI.10)}$$

$$\Delta w^{i}_{jkl} = \alpha \, (y_i^d - y_i^a) \, x_j \, x_k \, x_l \qquad \text{(VI.11)}$$

where Δw is the modification quantity for each weight value at each update step, α is the learning parameter(Reid used $\alpha = 1$ in his HONN), y_i^d denotes the desired i^{th} output, y_i^a denotes the actual i^{th} output, and x's are the inputs.

Using this learning rule, the weight values are updated iteratively until **all** the actual outputs are equal or close enough to the desired ones. Then the learning process is terminated and we say that the network has *converged*.

It was shown by Minsky [4] that if there exists a mapping between the input and output patterns, then the perceptron learning rule will find a solution(i.e. a set of proper weight values) in finite iterations. However, two points are worth mentioning here: first, it is not guaranteed that the perceptron learning rule will find a set of weights to realize **any** desired mapping; second, as stated by Minsky "All solution vectors (i.e. each solution vector stands for a set of weights) form a 'convex cone', and the program (the perceptron learning rule) will stop changing A (a certain solution vector) as soon as it penetrates the boundary of this cone". In other words, for a desired mapping which has solution(s), the perceptron learning rule cannot guarantee to find the "best one" (i.e. the set of weights implementing the optimal mapping

between the input and output patterns) in any sense. Therefore, even though the perceptron learning rule has the advantages of simplicity and rapid convergence rate (compare with the general *delta rule*, and the *Hebbian rule*), it is not sufficient to handle the real world pattern recognition applications in which the classification is difficult (any arbitrary mapping could be required) and the solution (set of weight values) is desired to be optimal.

Another simple learning mechanism used for ANN with one layer weights is the one use in associative memory (AM). It is referred as *Hebbian learning rule*. This learning is proposed based on the *conjunction theory* of learning [5]. A synapse strengthens if both the presynaptic and the postsynaptic neuron are active at the same time. In an ANN, the weights are the synapses between two layer of neurons, and the weight values are the synaptic strength. In a single layer, feed-forward ANN, the input layer neurons are the presynaptic neurons, and the output layer neurons are the postsynaptic neurons. The signal value of each neuron is its activity level. Unlike the *delta rule*, *Hebbian learning rule* can be expressed as:

$$\Delta m_{ij} = \alpha\, s_i\, f_j \qquad \text{(VI.12)}$$

where α is the learning parameter, the f_j is the j^{th} output neuron activity, and s_i is the i^{th} input neuron activity. Kohonen developed the AM based on this learning rule and has proven that the set of weight values obtained by AM represent "the optimal linear associative mapping" in the least square sense between **any** desired mapping between sets of input and output patterns [6]. In addition, this learning mechanism obtains all the weight values at once, no iteration is needed, so the training time is much shorter when compared with the delta rule (the speed difference is about one or two order of magnitudes as shown in section V of the *Software Test Report -- Simulations*).

We choose to use AM instead of the *delta rule* in the MHONN based on the theory and some encouraging simulation results obtained by Kohonen. In reference [7], after the AM was trained, it could correctly recognize testing patterns that experienced severe noise corruption and occlusion. These results indicates that the AM is quite robust to these types of image degradations.

The MHONN with AM (based on *Hebbian learning rule*) is illustrated in Fig. VI.4 In this figure, $\mathbf{f}$ denotes a desired output pattern (a vector), and f_j denotes the j^{th} component of the vector; $\mathbf{s}$ denotes a stimulus pattern (a vector) to the AM, and s_i represents the i^{th} component of the vector; $\mathbf{r}$ denotes the response vector (i.e. the output vector of the AM), and r_j is the j^{th} element of the vector. The $\mathbf{s}$ is the output pattern from the shift invariant accumulators. Let $\mathbf{M}$ denote the weight matrix with each element corresponding to an adjustable weight in the MHONN. The matrix $\mathbf{M}$ which implements the mapping between a set of inputs $\{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_k, \cdots, \mathbf{s}_p\}$ and a set of desired outputs $\{\mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_k, \cdots, \mathbf{f}_p\}$, must satisfy the following equation:

$$\mathbf{f}_k = \mathbf{M}\,\mathbf{s}_k \qquad \text{for } k = 1, 2, \cdots, p \qquad \text{(VI.13)}$$

The $\mathbf{M}$ is obtained by the following equation:

$$\mathbf{M} = \mathbf{F}\,\mathbf{S}^{+} \qquad \text{(V.14)}$$

where $\mathbf{F} = (\mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_k, \cdots, \mathbf{f}_p)$, $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_k, \cdots, \mathbf{s}_p)$, and $\mathbf{S}^{+}$ is the pseudoinverse of $\mathbf{S}$ [9]. If the stimulus vectors $\mathbf{s}_k$'s are linearly independent, then $\mathbf{S}^{+} = (\mathbf{S}^T\,\mathbf{S})^{-1}\,\mathbf{S}^T$.

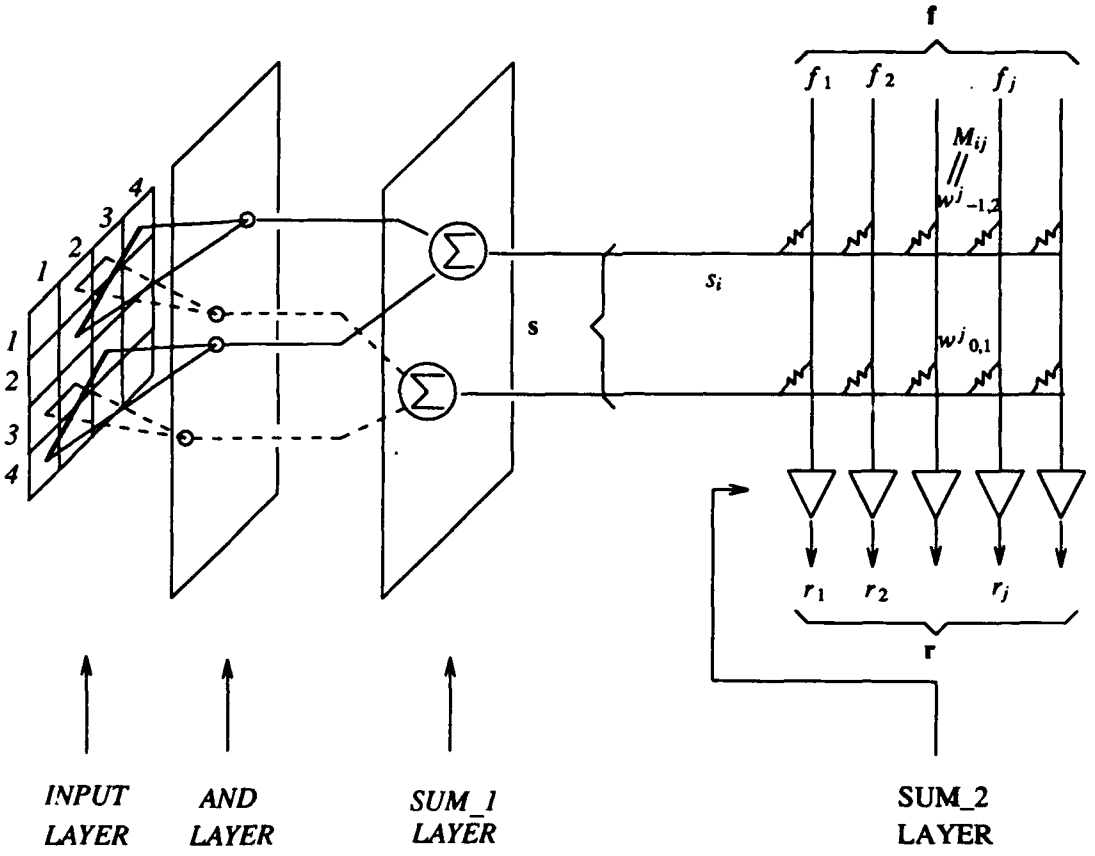


Figure VI.4 MHONN with associative recall learning mechanism

Simulation design and results using this approach are given in the Sect. V of the *Software Test Description -- Simulations* and *Software Test Report -- Simulations* of this report. In the following section we summarize simulation results.

4. SIMULATIONS

In this section we present the results of simulations performed with the MHONN subsystem. We not only did simulations with gray level images in the LSM computation plane, but also with gray level images in the rectangular image plane. Simulations with different types of noise, partial occlusions, etc. were also performed. A detailed description of all the simulations is given in the report *C. Software test- Simulations, Section VI.* Here we only give some significant simulation results.

Figure VI.5 shows the three ICBM images used for simulations. These are the original images. They were rotated, scaled and displaced in software to perform some of the simulations as explained below.

a. MHONN OS, CS Invariant Pattern Recognition with Gray-Level Input

Table VI.1 shows the recognition results when the shifted images are presented to the MHONN. In this table, the input patterns are listed in the first column, the recognition results are

given in the last column, values in the second column (called *energy*) give the number of non-zero intensity pixels in each input image, and values in the other middle columns are the MHONN output values. The notation "icbm1.u10.r-15" means that the pattern "icbm1" is shifted up by 10 rows, and cyclically shifted to the right by -15 columns (i.e., cyclically shifted to the left by 15 columns). If a certain pattern is misclassified, a "*" is marked in the table for attention. Recall that the simulations are done in the LSM computation plane in which vertical shifting is equivalent to scaling and horizontal shifting to rotation on the optical axis for the rectangular image.

From the simulation results we see that as long as no pixels are shifted out of the image (i.e. the case of the first two shifted patterns in each class in Table VI.1), the MHONN output values are the same as the desired output values. This indicates that the MHONN can perform **precise** OS and CS invariant pattern recognition for gray-level images.

The performance of MHONN (with multipliers) for gray-level input OS and CS invariant pattern recognition can be considered to be successful. When only OS and CS are applied to the input, the recognition results are correct. When small occlusions occur in the input, the MHONN can still correctly classify the patterns, but it fails when occlusion becomes severe. This simulation result shows the MHONN's potential to handle gray-level patterns, however, further studies are necessary to improve the MHONN's performance in this type of applications.

b. MHONN OS, CS Invariant Pattern Recognition with Binary Input

Simulation results of this test are given in Table VI.2. In Table V.2, all notations and table conventions are the same as those in the previous subsection. We see that each shifted pattern and its original pattern give exactly the same outputs. This indicates that the MHONN can perform **precise** shift invariant pattern recognition for binary images.

c. MHONN's Noise Tolerance

Table VI.1 MHONN OS, CS invariant pattern recognition with gray-level inputs

input	energy	O_1	O_2	O_3	recognition result
icbm1	282	**1.00**	0.00	0.00	icbm1
icbm1.u10.r-15	282	**1.00**	0.00	0.00	icbm1
icbm1.u-22.r22	282	**1.00**	0.00	0.00	icbm1
icbm1.u-27.r-13	243	-0.95	**2.19**	0.00	icbm2 *
icbm2	293	0.00	**1.00**	0.00	icbm2
icbm2.u13.r20	293	0.00	**1.00**	0.00	icbm2
icbm2.u-20.r-18	293	0.00	**1.00**	0.00	icbm2
icbm2.u26.r22	247	-1.82	**3.03**	0.00	icbm2
ic	630	0.00	0.00	**1.00**	icbm3
ic u14.r-8	630	0.00	0.00	**1.00**	icbm3
ic .u-5.r30	630	0.00	0.00	**1.00**	icbm3
ic u-25.r-8	622	-0.27	0.32	**1.00**	icbm3

Table VI.2. MHONN OS, CS invariant pattern recognition with binary inputs

input	energy	O_1	O_2	O_3	recognition result
icbm1	108	**1.000**	0.000	0.000	icbm1
icbm1.u8.r12	108	**1.000**	0.000	0.000	icbm1
icbm2	96	0.000	**1.000**	0.000	icbm2
icbm2.u-12.r17	96	0.000	**1.000**	0.000	icbm2
icbm3	196	0.000	0.000	**1.000**	icbm3
icbm3.u-7.r-18	196	0.000	0.000	**1.000**	icbm3

In this simulation, for a given SNR, five noisy patterns per class are used, and the network performance is measured by its *recognition rate* (RR) which is defined as the number of correctly recognized trials divided by the total number of trials tested for each class.

In Table VI.3 we see that with the increasing of noise, the O_1 value begins to decrease, while the O_2 begins to increase, O_3 and O_4 vary in a much smaller range, and stay close to zero. When noise level reach certain point, the peak value output element switches, and misclassification occurs. Another observation is that the noise tolerance of the MHONN is pattern dependent, for instance, at SNR = 0.75, the ICMB1 is recognized correctly, but the Ellipse is not. This phenomenon is very common with most of the pattern recognizers -- regardless of ANN based or conventional types. This is because the pattern recognizer forms an uneven division of the pattern space. In this case, apparently, the ICBM3 occupies a larger space than the other three classes, so when severe noise is added, all the patterns are classified as ICBM3. It is desirable for a pattern recognizer to divide the pattern space in a more uniform manner. However, research of this subject is beyond the scope of this project.

c. MHONN's Tolerance to Occlusions

Table VI.3 MHONN noise tolerance test result -- LSM icbm1 as input

SNR	O_1	O_2	O_3	O_4	recognition result
∞	**1.000**	0.000	0.000	0.000	icbm1
10	**0.982**	0.029	0.000	0.027	icbm1
2	**0.940**	0.211	-0.025	0.050	icbm1
1	**0.875**	0.458	-0.094	0.022	icbm1
0.75	**0.808**	0.740	-0.104	-0.047	icbm1
0.7	**0.820**	0.707	-0.123	0.020	icbm1
0.7	0.781	**0.799**	-0.109	-0.043	icbm3 *
0.65	**0.816**	0.788	-0.138	0.012	icbm1
0.65	0.763	**0.878**	-0.114	0.088	icbm3 *
0.6	0.750	**0.869**	-0.112	0.069	icbm3 *

The purpose of this simulation is to evaluate MHONN's tolerance to occlusion. Table VI.4(a)-(b) gives the testing results for the four classes of pattern. The table convention is the same as before. In the last column of each table, if a pattern is misclassified, a "*" will be marked; if a pattern is correctly recognized, but if the difference between the maximum output value and the second largest output value is less than 0.1, we say that the pattern is "marginally recognized", and a "?" will be marked.

From Table VI.4 we see that out of sixty-six cases have been tested, only one pattern -- "icbm1.0.55.0.64" is misclassified as trapezoid. In this case, only sixteen on-pixels, which is about 14.8% (16/108 = 14.8%, i.e. about 85% of the input pixels are deleted) of the original signal energy, are left in the patterns. In cases that patterns are marginally recognized, the testing patterns' signal energies are all less than 35.6% ("ellip.0.40.0.64") of the original ones. These can be considered as severe occlusions because about 2/3 of the signals are missing. Simulation results of this test show that the MHONN is very robust to occlusions in its input patterns. We want to emphasize that the occlusion situation discussed in this section is different from occlusion of the **sensed** (input) image **before** the LSM.

5. SUMMARY

In this section, we first pointed out the design goal and the position of the MHONN pattern recognizer in the overall system. Then the detailed MHONN design which include the architecture design, invariance algorithm development, and the associative mapping learning mechanisms were presented. This subsystem is designed to achieve OS, CS invariant pattern recognition in practical situations. Finally,the performance of MHONN was evaluated with respect to invariance in pattern recognition, noise tolerance, robustness to input pattern deformation and degradation. From the simulation results, we find that: (1) the MHONN with the OS, CS weight constraints, can perform OS, CS invariant robust pattern recognition for gray-level and binary input images, given that the objects are completely included in the window; (2) the MHONN has very high noise tolerance, it can correctly recognize patterns with noise signal almost as strong as the original information signal; (3) the MHONN can perform properly under severe input pattern occlusion.

Table VI.4(a) MHONN performance test result -- occlusion in LSM icbm1

input	energy	O_1	O_2	O_3	O_4	recog. result
icbm1	108	**1.000**	0.000	0.000	0.000	icbm1
icbm1.0.10.0.64	90	**0.689**	-0.008	0.014	0.013	icbm1
icbm1.10.20.0.64	91	**0.707**	-0.013	0.006	0.020	icbm1
icbm1.20.30.0.64	91	**0.703**	-0.013	-0.009	0.039	icbm1
icbm1.30.40.0.64	95	**0.765**	-0.012	0.010	0.022	icbm1
icbm1.40.50.0.64	91	**0.706**	-0.014	0.001	0.028	icbm1
icbm1.50.60.0.64	88	**0.659**	-0.009	-0.008	0.031	icbm1
icbm1.0.20.0.64	73	**0.451**	-0.026	0.005	0.047	icbm1
icbm1.20.40.0.64	78	**0.510**	-0.026	-0.012	0.067	icbm1
icbm1.0.30.0.64	56	**0.262**	-0.037	-0.050	0.108	icbm1
icbm1.20.50.0.64	61	**0.308**	-0.038	-0.053	0.116	icbm1
icbm1.0.40.0.64	43	**0.149**	-0.034	-0.017	0.080	icbm1
icbm1.0.50.0.64	26	**0.049**	-0.021	0.005	0.044	icbm1 ?
icbm1.0.55.0.64	16	0.017	-0.012	0.007	**0.023**	trap *
icbm1.0.64.0.35	48	**0.276**	-0.016	0.003	-0.032	icbm1

Table VI.4(b) MHONN performance test result -- occlusion in LSM icbm3

input	energy	O_1	O_2	O_3	O_4	recog. result
icbm3	196	0.000	**1.000**	0.000	0.000	icbm3
icbm3.0.10.0.64	159	0.013	**0.580**	0.062	0.138	icbm3
icbm3.10.20.0.64	163	-0.010	**0.719**	0.001	-0.043	icbm3
icbm3.20.30.0.64	173	-0.023	**0.882**	-0.025	-0.132	icbm3
icbm3.30.40.0.64	172	0.015	**0.870**	-0.073	-0.098	icbm3
icbm3.40.50.0.64	175	-0.030	**0.875**	-0.050	-0.076	icbm3
icbm3.50.60.0.64	139	0.016	**0.409**	0.076	0.129	icbm3
icbm3.0.20.0.64	126	-0.002	**0.346**	0.062	0.097	icbm3
icbm3.20.40.0.64	149	-0.006	**0.762**	-0.079	-0.215	icbm3
icbm3.0.30.0.64	103	-0.016	**0.280**	0.017	0.013	icbm3
icbm3.20.50.0.64	128	-0.024	**0.682**	-0.133	-0.278	icbm3
icbm3.0.40.0.64	79	-0.004	**0.230**	-0.033	-0.052	icbm3
icbm3.0.50.0.64	58	-0.010	**0.198**	-0.062	-0.098	icbm3
icbm3.0.55.0.64	36	-0.004	**0.123**	-0.043	-0.081	icbm3
icbm3.0.64.0.20	190	0.003	**0.915**	0.030	0.055	icbm3
icbm3.0.64.0.25	180	0.018	**0.756**	0.055	0.165	icbm3
icbm3.0.64.0.30	160	0.060	**0.564**	0.032	0.198	icbm3
icbm3.0.64.0.35	114	0.097	**0.255**	0.007	0.143	icbm3
icbm3.0.64.0.40	50	0.012	**0.083**	-0.021	0.010	icbm3 ?
icbm3.0.64.0.45	28	0.003	**0.036**	-0.010	0.003	icbm3 ?
icbm3.0.64.0.50	14	0.003	**0.006**	0.000	0.004	icbm3 ?

SECTION VII

LINE CORRELATOR TRACKER FOR SCALING AND ROTATION

1. INTRODUCTION

The line-correlator-target-tracker reported in this section can be used to determine changes in target size and orientation when used with the LSM computation plane image. As the target gets closer to the interceptor, its image increases in size. If the initial distance is known, the change in size determines the distance. If either the target or the seeker rotate with respect to the optical axis, the LCT is also capable of determining the degree of rotation. The images used by the LCT are in the *computation plane* of the *logarithmic-spiral mapping* .

2. LINE CORRELATOR TRACKER

An original one dimensional correlation tracker for motion prediction was developed and used for rotation and scaling motion prediction in the log-spiral computation plane. The algorithm is called "line correlator tracker" (LCT). It has recursive, spatio-temporal, correlation characteristics and also possesses simplicity and separability properties. These properties make the design and implementation of the algorithm possible in a highly parallel fashion using neural networks. The most relevant characteristic of the algorithm is that the motion prediction problem is solved without using correspondence. Several approaches have been previously proposed to solve the motion prediction problem without using correspondence [1,2]. The approach proposed in this section is different and produces good experimental results with relatively few computations.

Consider a dynamic scene. In general, the intensity of the light reflected by the scene will be a function of location and time, $I(x,y,t)$. We can define its gradient in this 3D space,

$$\nabla I = \left[\frac{\partial I}{\partial x} \quad \frac{\partial I}{\partial y} \quad \frac{\partial I}{\partial t}\right]^T \tag{VII.1}$$

and its gradient in 2D geometric space,

$$\nabla_{\tilde{x}} I = \left[\frac{\partial I}{\partial x} \quad \frac{\partial I}{\partial y}\right]^T \tag{VII.2}$$

For uniform illumination, changes in intensity at a point are due to object motion (assume that other disturbances are inhibited). If the intensity of a point in the object does not change with respect to s (in other words, an infinitesimal spatial displacement corresponds to a change dt in time between consecutive frames), then $\frac{dI}{ds} = 0$. Let the unit vector $\hat{u}$ be given by

$$\hat{u} = c\,(\vec{v}(\tilde{x},t), 1) \tag{VII.3}$$

where $c = (|v|^2 + 1)^{-\frac{1}{2}}$ and $\vec{v}(\tilde{x},t)$ is the point velocity. Then, using the above, we have

$$\vec{v} \cdot \nabla_{\tilde{x}} I + \frac{\partial I}{\partial t} = 0 \tag{VII.4}$$

Let τ be the time between two successive frames, $\tau = t_2 - t_1$. We then have

$$\begin{aligned}\Delta(\tilde{x},t) &= \tau\vec{v}(\tilde{x},t) \\ &= (\Delta x\ \Delta y)^T\end{aligned} \tag{VII.5}$$

then

$$\frac{\partial I}{\partial t} \approx \frac{I(\tilde{x},t) - I(\tilde{x},t-\tau)}{\tau} \tag{VII.6}$$

The approximation of (*VII.* 6) is accurate as long as the object motion is relatively small. Using finite increments in (*VII.* 12) ,

$$I(\tilde{x},t_1) = I(\tilde{x},t_2) + \frac{\partial I(\tilde{x},t_2)}{\partial x}\Delta x + \frac{\partial I(\tilde{x},t_2)}{\partial y}\Delta y \tag{VII.7}$$

In order to deal with large object motion, a recursive process is essential. Thus, to obtain an iterative algorithm, we proceed as follows:

Let's consider only one-dimensional motion only for the moment. Rewrite (*VII.* 7) as

$$I(x_o,t_1) = I(x,t_2)\Big|_{x=x_j} + \frac{\partial I(x,t_2)}{\partial x_j}\Big|_{x=x_j}\Delta x_j \tag{VII.8}$$

where

$$\{R_j\} = Sampled\Big[I(x_o,t_1)\Big]$$

$$\{T_j\} = Sampled\Big[I(x,t_2)\Big|_{x=x_j}\Big]$$

and x_o is an initial pixel position in the first frame; x_j is an arbitrary chosen pixel position in the second frame. Normally, we will choose x_j to be equal to x_o unless a priori information about x_j is provided. Thus, for any pixel and for a discrete case, we have

$$\{R_j\} = \{T_j\} + \frac{\partial I(x,t_2)}{\partial x_j}\Big|_{x=x_j}\Delta x_j \tag{VII.9}$$

where

$$\frac{\partial I(x_j,t_2)}{\partial x_j} \approx T_{j+1} - T_j$$

Now, define the correlation of $\{R_j\}$ and $\{T_j\}$ as

$$C = \sum_{j=1}^{m} R_j T_j \tag{VII.10}$$

Combine (*VII.* 9) and (*VII.* 10) together and for $\Delta x_i = \Delta x_j$ for all j,yield

$$\Delta x_i = \frac{\sum_{j=1}^{m} R_j^2 - \sum_{j=1}^{m} R_j T_j}{\sum_{j=1}^{m} R_j \frac{\partial I}{\partial x_j}} \tag{VII.11}$$

If motion would occur in the x direction only, then any row in the image would produce a good estimate. However, motion can generally occur in both the x and y directions (in computation LSM plane, i.e., scaling and rotation of the object in image plane).

The problem for general x/y translation is that corresponding rows (and columns) in frames "i" and "$i+1$" within the moving area will be shifted with respect to each other. For example, if $\Delta x = 3$ pixels and $\Delta y = 5$ pixels, the object has moved right by 3 pixels and up by 5 pixels and rows in frame $i+1$ will be at location $k+5$ with respect to corresponding rows in frame i (where k is row number). Hence, if a search is performed for each row in both the positive and the negative directions to find the best matching row in the second frame, this problem will be solved. The question remains: how many rows must comprise the search area? This, of course, will depend on how large a motion will occur between frames which, in turn, depends on sampling rate and object speed. A reasonable figure is to allow for displacements of at most ten percent of the maximum object dimension in pixels.

The row under investigation is correlated with all the rows in the search region and a pixel by pixel estimate is performed. The row which produces the most consistent estimate is likely to be the matching one. This can be best illustrated by Fig.VII.1, where we search 3 rows above and 3 rows below the *kth* row in the first frame. If the object, for example, moves up 2 pixels, then the $(k-2)th$ row in the second frame is perfectly matched to the *kth* row in the first frame. That is, the estimate obtained from correlating these two rows using (*VII.* 11) is a consistent estimate. Other matches will produce inconsistent results.

3. LCT NEURAL NETWORK IMPLEMENTATION

The LCT algorithm can be implemented by means of neural network models currently available. We have chosen a Hopfield-Tank network because its analog VLSI implementation is possible with current technology.

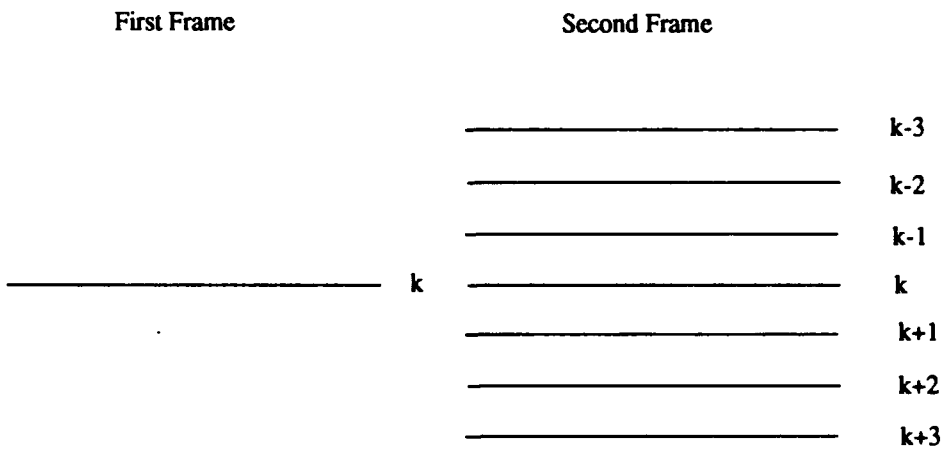


Figure VII.1. Row Search Mechanisms

The network is a two-layer HT network. The first layer consists of $(2m + 1)$ "planes". (Plane, is used here in the HT sense, **not** in the sensor sense (as in "image plane")). Each plane contains n HT networks and each network has $(2D + 1)$ neurons, where D is the maximum row displacement. Each network computes row displacements and feeds its output directly to the second layer. This layer computes the overall displacement based on the outputs of the first layer. There is a single plane in the second layer. The plane contains $(2m + 1)\times(2D + 1)$ neurons. The outputs from the second layer are the horizontal and vertical displacements. Figure VII.2 depicts the general architecture of the HT network. The following subsections describe the energy functions of the two layers.

a. First LCT Neural Network Layer

The model in the first layer contains binary neurons representing the row displacement between the two images (i.e., displacement along the abscissa). We use $n\times(2m + 1)\times(2D + 1)$ neurons. For implementation, we discretize the row displacement by letting $-D \le j \le +D$. We also let $[V_j]_{lk}$ represent the state of the jth neuron of lth row and kth plane. When the neuron $[V_j]_{lk}$ is 1, it means that the row displacement for the lth row in the kth plane is j. If subpixel accuracy is desired, one can simply increase the number of neurons within the tracking window. If, for example, five neurons are used per pixel, an accuracy of 0.2 pixel will be achieved. For this model to function, an energy function must be developed such that only one neuron within each row is turned on when the network reaches stable state. The energy function for the lth row of kth plane is given below.

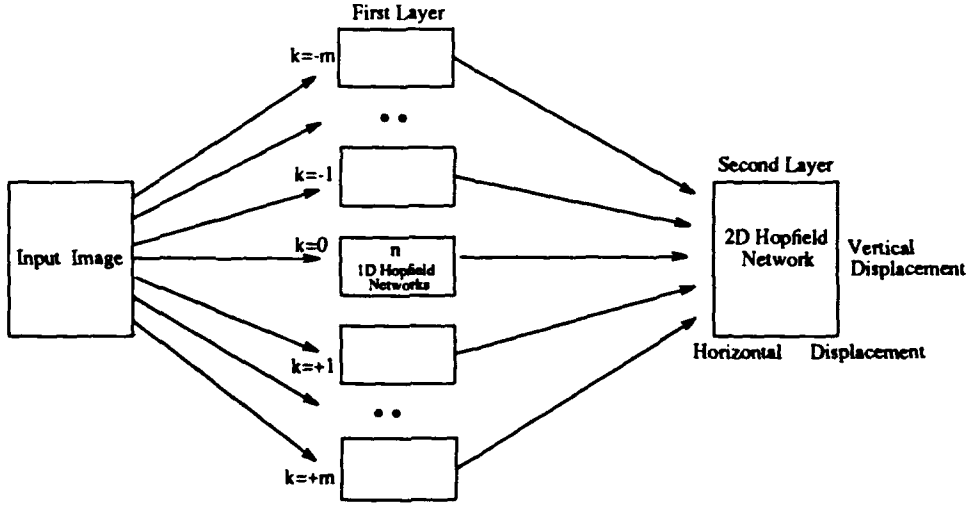


Figure VII.2. General Architecture of the Neural Network

$$E_{lk} = \frac{A}{2}\left[\sum_i\sum_{j\neq i}\left[V_i\right]_{lk}\left[V_j\right]_{lk} - \sum_i\left[V_i\right]_{lk}\right] + \frac{B}{2}\left[\sum_j\left[V_j\right]_{lk} - 1\right]^2 + C'\sum_j\left[S_{lk} - Q_{lk}i\right]^2\left[V_j\right]_{lk} \tag{VII.12}$$

where

$$S_{lk} = \sum_j R_{lj}^2 - \sum_j R_{lj}T_{(l+k)j}$$

$$Q_{lk} = \sum_j \frac{\partial I}{\partial x_{(l+k)j}} R_{lj}$$

$$C' = \frac{C}{(i+\varepsilon)^2}$$

The first and the second terms in (*VII.* 12) provide row inhibition and global inhibition, respectively. These two terms assure that there is one and only one neuron "on" for the *l*th row of the *k*th plane when the network reaches the stable state. These two terms are also known as the constraint terms. The last term is the data term or the objective term. Without this term, a neuron will be on randomly. The presence of this term will force the neuron which corresponds to the row displacement, to turn on. Notice that the third term is taken directly from (VII.11), with the *i* parameter corresponding to Δx_{lk}. Due to the poor scaling property of the HT network, the constant *C* is scaled to sensitize the data to row displacement *i*. ε ($0 < \varepsilon < 1$) is added to the scaling term to avoid dividing by zero in case of zero motion. Rearranging (VII.12), the first term is written

$$-\frac{1}{2}\sum_i\sum_{j\neq i}(-A(1-\delta_{ij}))\left[V_i\right]_{lk}\left[V_j\right]_{lk} - \sum_i\frac{A}{2}\left[V_i\right]_{lk} \tag{VII.13}$$

the second term,

$$-\frac{1}{2}\sum_i\sum_j(-B)\left[V_i\right]_{lk}\left[V_j\right]_{lk} - \sum_i B\left[V_i\right]_{lk} + 1 \tag{VII.14}$$

and the third term,

$$-\sum_j(-C')\left[S_{lk} - Q_{lk}\,i\right]^2\left[V_j\right]_{lk} \tag{VII.15}$$

where δ_{ij} denotes Kronecker delta. The general energy function of a HT network [3,4] can be written as

$$E_{lk} = -\frac{1}{2}\sum_i\sum_j\left[T_{ij}\right]_{lk}\left[V_i\right]_{lk}\left[V_j\right]_{lk} - \sum_i\left[I_i\right]_{lk}\left[V_i\right]_{lk} \tag{VII.16}$$

By comparing the terms in (VII.13,14,15) with the corresponding terms in (VII.16), we determine the interconnection strengths (connection matrix, $[T_{ij}]_{lk}$ where *T* is used for "transconductance") and the bias inputs (excitation term, $[I_i]_{lk}$ where *I* is used for "current") as

$$\left[T_{ij}\right]_{lk} = -A(1-\delta_{ij}) - B$$
$$\left[I_i\right]_{lk} = \frac{A}{2} + B - C'\left[S_{lk} - Q_{lk}\, i\right]^2 \qquad \text{(VII.17)}$$

Notice that the quadratic terms in the energy function define a connection matrix and the linear terms define input bias current.

b. Second LCT Network Layer

The second layer contains $(2m+1)\times(2D+1)$ binary neurons representing the overall row and column displacements between the two images. The horizontal displacement is discretized by letting $-D \le j \le +D$. The vertical displacement is represented by index k. V_{kj} represents the state of the kjth neuron. When V_{kj} is 1, the horizontal displacement is j and the vertical displacement is k. When the network reaches stable state, there should be only one neuron on for the entire layer. Since this is a 2D problem, a 4-dimensional energy function is required. Using p and q subscripts for row indices and r and s as the column indices, such an energy function is given by

$$E = \frac{A}{2}\sum_p\sum_r\sum_q\sum_s V_{xi}V_{y_j}\left[1-\delta_{pr,qs}\right] + \frac{B}{2}\left[\sum_p\sum_r Vpr - 1\right]^2 - C\sum_p\sum_r\left[\sum_l V'_{lr}\right]_p V_{pr} \quad \text{(VII.18)}$$

The first term provides row and column inhibitions, and the second term global inhibition, thus assuring that only one neuron is turned on for the entire layer. The last term forces the neuron corresponding to the object translation to turn on. The V'_l is the neuron output at the lth row of the first layer. By summing up the neurons from the different rows at the same column for a given kth plane (corresponding to xth row in the second layer), we can provide proper excitation for each neuron in the particular location. For example, if l neurons in the jth column of the kth plane in the first layer are all 1's or nearly all 1's (few 0's), then the output of the neurons in this column should have a minimal standard deviation. Thus, it is appropriate to add the sum of all neurons in this column to the excitation term at pr position, where p indicates the corresponding kth plane and r is the index to the column which corresponds to the most consistent estimate (the column whose standard deviation is minimal). The general energy function for 4-dimensional problems can be written as

$$E = -\frac{1}{2}\sum_p\sum_s\sum_r\sum_q T_{pr,qs}V_{pr}V_{qs} - \sum_p\sum_r I_{pr}V_{pr} \qquad \text{(VII.19)}$$

Through similar analysis as in the previous section, the weight connection matrix and the bias input are derived.

$$T_{pr,qs} = -A\left[1-\delta_{pr,qs}\right] - B$$
$$I_{pr} = B + C\left[\sum_l V'_{lp}\right]_p \qquad \text{(VII.20)}$$

As can be seen, the weight connection matrices for both layers are fixed and independent of image sequence frames. Thus, once the weight matrices are set, they can be used for the entire tracking task. In addition, the constant A, B, and C for both layers are also insensitive to the image sequence frames. Motion estimation is carried out by a neuron evaluation. Each neuron

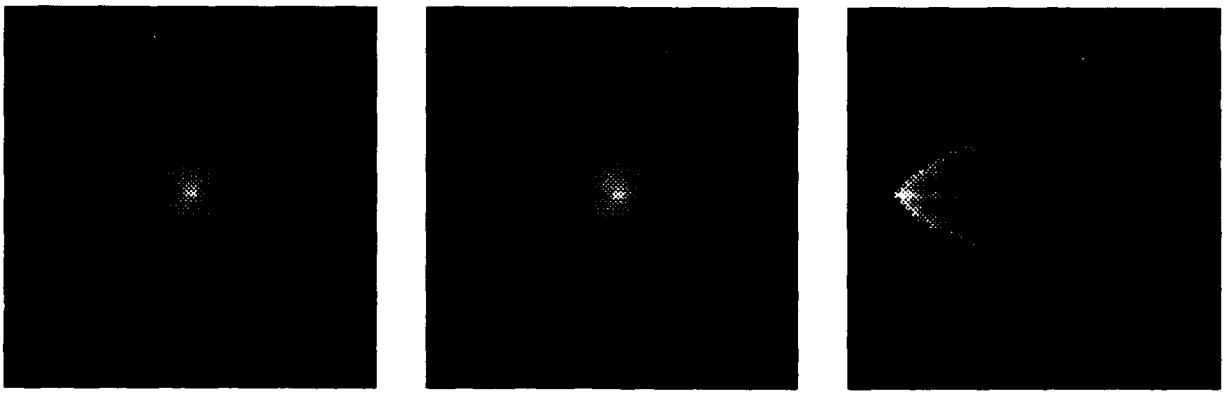
asynchronously evaluates its state and readjusts itself according to the sigmoid function. The network proposed here calculates the motion based on each individual row of the image. Thus, the size of the network representing the row is relatively small. This increases the convergent probability of the network. In addition, the output of neurons is not sensitive to their initial states. The simulation results shown in the simulation section support the claims.

4.LCT SIMULATIONS

The ability of the lCT to track target translations was examined using the 90 degree plume image of Fig.VII.3 . A 256x256 image was used without segmentation, resulting in a relatively large tracking window. The object was translated in the image artificially to enable an accurate assessment of the LCT capability. The translation results, contained in Table VI.1, illustrate that the LCT accurately tracks the target. The erroneous results (denoted by ****) indicate the maximum translation distance allowable. Likewise, the ability of the LCT to track target rotations and scalings is shown in Table VI.3 . For this table, the input to the LCT was from the LSM computation plane. The results are, again, very good for reasonable values of the scaling and rotation.

Table VII.1. LCT Translation Results for 90 Degree Plume

Actual Displacement		Estimated Displacement	
Δx	Δy	Δx	Δy
3.0	5.0	2.817	4.768
-10.0	-4.0	-10.321	-4.279
15.0	-7.0	17.32	-7.351
9.0	-25.0	****	****



a. 0 Degree Missile Plume **b. 15 Degree Missile Plume** **c. 90 Degree Missile Plume**

Figure VII.3 ICBM Plume Images

Table VII.2. LCT Rotation/Scaling Results for 90 Degree Plume			
Actual		Estimated	
$\Delta\theta$ (rotation in degrees)	Δk (scale)	$\Delta\theta$ (rotation in degrees)	Δk (scale)
-7.5	1.00	-7.30	1.000
-11.0	0.72	10.69	0.672
-16.0	0.50	****	****

SECTION VIII

SINGLE PIXEL TARGET DETECTION

1. INTRODUCTION

Triple-indexed data is defined to represent a set of images, of which two indices are spatial coordinates (x, y), and the third is the time coordinate or discrete frame number. This is often referred to as 3-D image or a time sequence. When a target of small size (< 10 m in length) is remote from the sensor (> 100 km), it is imaged at only one pixel or less in an image frame. This type of target is referred to as a "pixel-sized" target. A time sequence of images containing a pixel-sized target trajectory which intersects each frame at only one pixel is sampled by a sensor. The position and the velocity of the target is unknown and the trajectory is arbitrary. Multiple targets may also be present in the same sequence. Such image sequences can be obtained from a space-borne sensor mounted on satellites or "smart" target seekers when, for example, an ICBM releases warheads and decoys. For the task of intersecting the targets in mid-flight, pixel-sized target detection and tracking in real time is of essential importance.

The sampled image sequence contains randomly drifting background clutter and may also be contaminated by random sensor noise. The intensities of isolated noise pixels can become significantly higher than that of the target pixel. The difficulties of the detection and tracking task are clear: For pixel-sized targets, conventional pattern recognition methods fail for lack of shape information; there is no spatially high concentration of intensities to detect around the target due to the same reason; some randomly distributed high-intensity noise pixels have the same appearance as the targets in a frame. With little knowledge about the trajectories in the time sequence, the task becomes extremely difficult.

This section presents a new pipeline method for detection and imaging of pixel-sized moving targets. The *Pipeline Target Detection Algorithm* (PTDA) detects targets with arbitrary trajectories in a time sequence of images and simultaneously produces an image of the trajectories. The sampled image sequence is corrupted with randomly drifting background clutter as well as random sensor noise.

2. THE PIPELINE TARGET DETECTION ALGORITHM

With a pixel-sized target trajectory contained in the time sequence described in Sect.1, the only information for the detection lies in the trajectory continuity. The PTDA makes use of the spatial consistency of intensity of a target within a short time period, resulting from the continuous 3-D trajectory, to detect the existence of the target. Therefore, a necessary condition for proper algorithm performance is: *The trajectory of a target must be continuous and smooth.* Under this assumption, the target pixel can not make a big leap between two adjacent frames with a proper sampling rate of the sensor. Along the temporal axis, the target pixel travels a short distance in the spatial coordinates, one (or even a fraction of a) pixel, for example, at each sampling cycle. The PTDA is able to distinguish the regular distributions of target trajectories from the random distribution of noise, and to detect those of targets.

When a sequence of a few adjacent frames is accumulated and a column of small windows equal to the number of frames is applied to each pixel location, as shown in Fig.VIII.1, the distribution of pixel intensities within the windows is very different at different frame locations. When a trajectory segment is contained in the windows, the distribution of intensities is very consistent and regularly shaped. However, when the window column is at a location away from a target trajectory, very few consistent high-intensity pixels can be seen in the space confined to the windows and distribution is very irregular, if there is any. On the other hand, if one tries to inspect the intensity distribution of the complete frames, little significance can be observed. Based on this analysis, the PTDA focuses on a small neighborhood of a pixel for only a few frames at a time, and detects the existence of a target by seeking consistency of pixel intensity distribution within the confined 3-D space. With a sample sequence of images from the sensor, the pipeline target detection system forms a pipe that consists of a few frames (typically three to five), and pushes the time sequence of images one frame per step through the pipeline. At each cycle, a column of windows (normally square-shaped) equal to the number of frames in the pipeline, is centered on each pixel of the image frames in the pipeline, in parallel. The target presence within a window column should result in a higher intensity value than in the surroundings, or than some threshold. This is detected by temporally summing the pixel intensities inside the window column. When this "high concentration" of intensity occurs, the computation of intensity centroid in the spatial-temporal sense will uniquely determine the center of the trajectory segment inside the space confined to the windows. The PTDA detects one trajectory pixel per cycle, and tracks the entire trajectory inside the time sequence at the sampling rate.

a. The Pipeline Structure

- *Pipeline*
  A fixed length **FIFO** (first-in-first-out) set of **n** $N \times N$ image frames, and a two-dimensional array of processing elements (PEs) form the pipeline. The dimension of the array is $N \times N$, and there is a PE at every pixel location. At each frame cycle, a frame is discarded from the bottom of the **FIFO** and a new one added to the top. This process is referred to as "updating the pipe".
- *Processing Element (PE)*
  A PE is a local processor for one pixel location capable of the following functions: 1) algebraic and logical functions; 2) image pixel value storage; 3) neighborhood connections and communications in the space defined by the Temporal Window Column.
- *Temporal Window Column (TWC)*
  A TWC is a column of n windows of dimension $w \times l$ (generally $w = l$) in the pipeline that are always centered in the same positions in each frame of the pipe. There is a window for every pixel of a frame in the pipeline, and n, the number of windows in the column for a given position (i.e. pixel), is equal to the number of frames in the pipeline.

b. The Continuity Filter

One of the major difficulties to overcome is to reduce the effect of noise in the time sequence. When some noise pixels with high intensity levels are distributed close together both spatially and temporally in the pipeline, their effect on the temporal window summation is significant and can result in false detection by the algorithm. Notice that these pixels do not have to be continuously distributed in adjacent frames for their contribution to the PE operation.

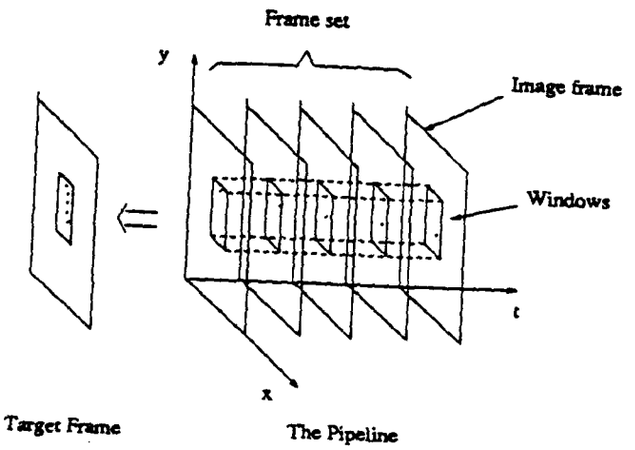


Figure VIII.1 The window-centroid method

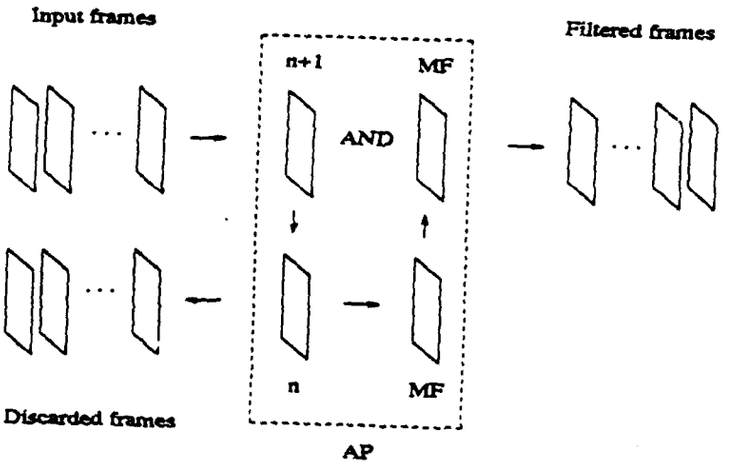


Figure VIII.2 AND-Pipe operations

In order to distinguish these noise pixels from target pixels, the property of temporal continuity of target trajectories has to be considered. The continuity and smoothness constraint of target trajectory restricts the pixels in adjacent frames to stay within a small neighborhood of each other. In other words, if there is a target pixel in the ith frame, the (i+1)th frame should also have a target pixel spatially close to the one in the ith frame. On the other hand, it is not very likely for two high-intensity noise pixels in a pair of adjacent frames to have such a property. The *Continuity Filter* (CF) is designed to use the continuity property of the target pixels in adjacent frames and the randomness of noise pixels to filter out the noise through the A-AND operation.

We define an *Analog-AND (A-AND)* function for grey-level algebraic operations. A function f(g(.), A), where g(.) is a grey-level function and A $\in$ {0, 1}, is define as A-AND, if it satisfies

$$f(g(.), A) = \begin{cases} g(.) & \text{if } A = 1 \\ 0 & \text{if } A = 0 \end{cases} \qquad \text{(VIII.1)}$$

The CF consists of a pipeline of two frames, called *AND Pipe* (AP). After initially pre-filling the AP, the CF operates in the following fashion (Fig. VIII.2):

1. Update the AP
2. Generate local masks: In **parallel** for every pixel in the bottom frame of the AP, if there is an "on" pixel, i.e., the intensity is higher than some threshold (determined by possible target intensities), set the corresponding position and pixels in a certain neighbor area (e.g. 3x3 area) in a *mask frame* (MF) to logic "1"
3. *A-AND* each pixel of the MF in **parallel** with the corresponding grey-level pixel in the top frame of the AP, and so generate a filtered frame (with grey-level intensity)

The CF produces one filtered image frame per frame cycle from the bottom of the AP in the pipeline fashion. As result of the filtering, target trajectory pixels remain unchanged in the sequence because of the consistency of the distribution. Most noise pixels, on the other hand, are filtered out as long as the high intensity-level noise pixels do not stay consistently in the same areas in adjacent frames. Notice that the filtered frames are still grey-level images with most of the frame area having intensity level 0 except for some isolated blocks around target pixels.

c. The Algorithm

I. Construct a *Test Pipe (TP)* of a sequence of n image frames, an *AND Pipe (AP)* of two image frames, and a single blank frame called *Target Frame (TF)*, Fig.VIII.3

II. Initialize the AP in two frame cycles by adding a frame to the top at each cycle

III. Initialize the TP. In **n** time steps,

   1. Update the AP
   2. Apply the *Continuity filter* in the AP as described above
   3. Add a frame from the output of the AP to the top of the TP

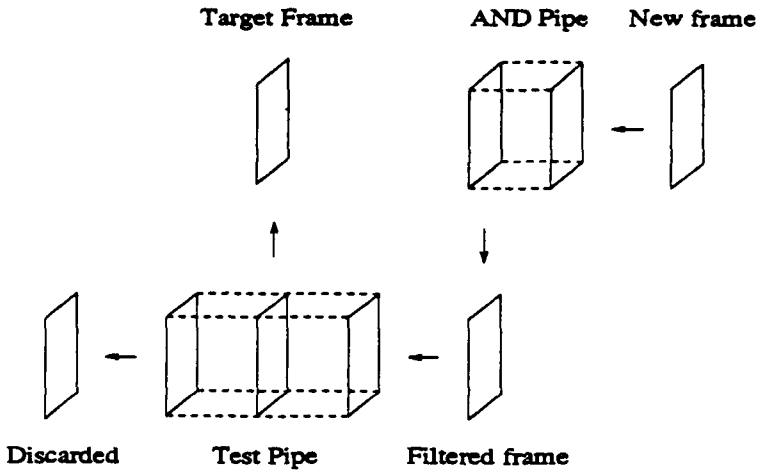


Figure VIII.3 *Operation flow of the PTDS algorithm*

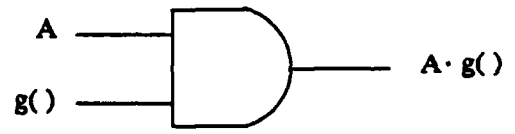


Figure VIII.4 A-AND function

IV. At each frame cycle, (refer to Fig. 4)

1. (*a*) Update the AP
   (*b*) Apply the CF in the AP

2. Update the TP by adding the output frame of the AP to the top of the TP

3. At each local PE in *parallel*,
   (*a*) Sum the pixel intensities inside 3-D space defined by the TWC:

$$A_i = \sum_{k=1}^{n} \sum_{x=-w/2}^{w/2} \sum_{y=-l/2}^{l/2} p(X_i + x,\ Y_i + y,\ k) \qquad \text{(VIII.2)}$$

where (X_i, Y_i) are the center coordinates of TWC at time i, and n, w and l are the dimension of the TWC in the Test Pipe
(*b*) If the sum is greater than a threshold (determined by possible target intensities), then go to step 4, else go back to step 1 for the next time step.

4. Compute the intensity centroid $(\hat{x}, \hat{y})$ in **parallel** in the PEs that have detected high intensity values at time step i:

$$\hat{x}_i = \frac{1}{A_i} \sum_{k=1}^{n} \sum_{x=-w/2}^{w/2} \sum_{y=-l/2}^{l/2} p(X_i + x,\ Y_i + y,\ k)(X_i + x) \qquad \text{(VIII.3.a)}$$

$$\hat{y}_i = \frac{1}{A_i} \sum_{k=1}^{n} \sum_{x=-w/2}^{w/2} \sum_{y=-l/2}^{l/2} p(X_i + x,\ Y_i + y,\ k)(Y_i + y) \qquad \text{(VIII.3.b)}$$

Record the centroids in the TF

5. For detected intensity centroid $(\hat{x}_i, \hat{y}_i)$ in the TF, find Euclidean distance between the current position and the previous time position, and compare it against a threshold:

$$\begin{cases} \text{if } \| \hat{x}_i - \hat{x}_{i-1} \| \le \Delta\hat{X} & (\hat{x}_i, \hat{y}_i) \textit{ accepted} \\ \text{and } \| \hat{y}_i - \hat{y}_{i-1} \| \le \Delta\hat{Y} & \textit{to be the trajectory pixel} \\ \textit{else} & (\hat{x}_i, \hat{y}_i) \textit{ not accepted} \end{cases} \qquad \text{(VIII.4)}$$

6. The instantaneous velocity of the target can also be determined by

$$\Delta x = \hat{x}_i - \hat{x}_{i-1} \quad \text{and} \quad \Delta y = \hat{y}_i - \hat{y}_{i-1} \qquad \text{(VIII.5)}$$

In the algorithm, steps I to IV.4 directly involve the detection of trajectories. As a consequence, possible target trajectories are reconstructed and tracked in the single frame TF. Step IV.5 is an additional means of keeping track of the multiple target situation, so that the system

knows at any time the number of candidate trajectories currently detected. The result of step IV.6 provides useful information for tracking. Notice that the steps of the algorithm are executed at each PE of the pipelines simultaneously. The PTDA eliminates slowly drifting background clutter and most of the random sensor noise effect and detects and tracks target trajectories of arbitrary shapes. Stationary and directly oncoming objects are recorded by the algorithm stationaryly (equivalent to a straight line trajectory parallel to the time axis). These "trajectories," hence, are fixed at one point in the Target Frame.

If a conventional pattern recognition algorithm such as the Hough transform is applied to the TF, the types of trajectories can be easily identified, e.g., straight lines, parabolas, etc. An ANN implementation of the Hough transform has been developed to apply to this algorithm as the trajectory recognition stage. Based on the result of this recognition compared with prior knowledge of the target trajectory (through target trajectory analysis), decisions can be made on whether or not a trajectory is of interest.

3. IMPLEMENTATION CONSIDERATIONS

One of the most important characteristics of the PTDA is its global parallelism. At each sampling cycle, a new frame from the sensor is fed to the top of the AP, and at the same time, a used frame is taken out from its bottom. The CF is then in process. Simultaneously, a filtered frame produced by AP from the previous cycle is input to the top of TP, and at the bottom a frame is discarded. The remaining steps of the PTDA are then applied. The algorithm tracks one pixel per cycle on the trajectories except for the first few cycles needed to pre-fill the pipes. All operations of the algorithm are also parallel-distributed in local PEs, but globally synchronized. Each operation of a PE is a localized simple operation (confined to a small 3-D space defined by the TWC), but synchronized in parallel with the same operation of the neighboring elements. Therefore, each operational cycle is expected to be very short and compatible with the sampling rate of the sensor. The parallelism of the algorithms ensures the timing requirement of real-time tracking tasks.

The PTDA can be implemented by any parallel distributed architecture. Good examples are the *Pyramid* type structure and the *Connection Machine* structure which have neighborhood operation capabilities. *Artificial Neural Networks* (ANNs) are also a good possibility for implementation of the algorithm. In fact, a *back propagation network* implementation of the algorithm has been developed by the authors with promising preliminary results.

4. SIMULATIONS

Simulations have been done with synthetic target trajectories in real image sequences. The original sequence contains slowly drifting background clutter, as shown in Fig.VIII.5(a). Two pixel-sized trajectories were generated and injected into the sequence. One of the targets moves in a 3-D linear fashion, while the other follows a 3-D parabolic trajectory. Two types of pseudorandom noise were then superimposed to the sequence. Pepper-and-salt type noise with saturated pixel intensity (255) was generated at random positions of uniform distribution in a frame. Different noisy sequences were obtained by adding different number of noise pixels to each frame. The reason why the intensity of noise pixels was set to a maximum value that outnumbered the target intensity was to exaggerate the noise effect and to simulate situations in which the images are dominated by noise intensities and the target is barely detectable. Gaussian pseudorandom noise was also generated and superimposed to each pixel location in each frame. Different variance values of Gaussian function specified noise levels of the noisy

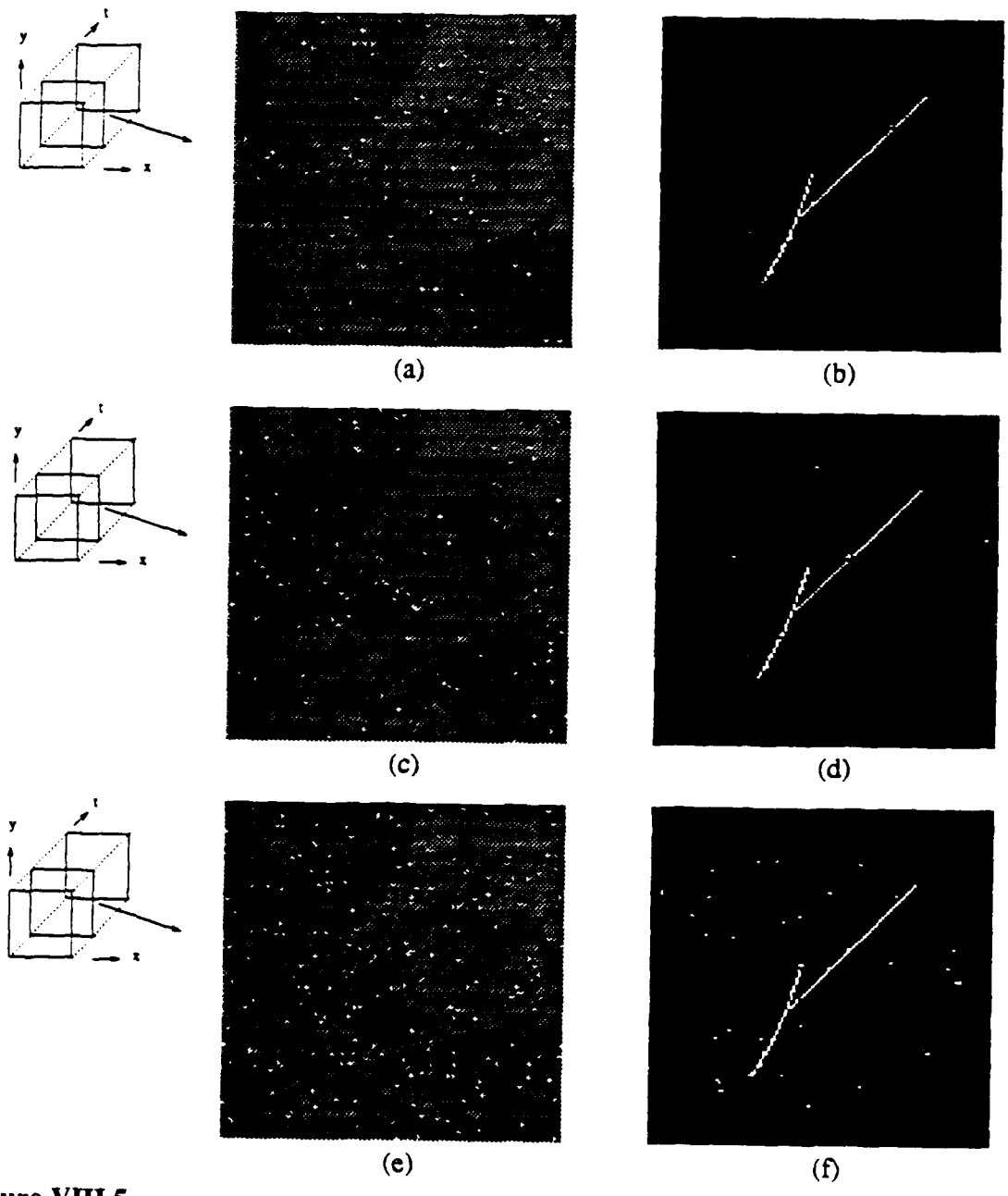


Figure VIII.5 Simulation results by the original system architecture with salt-noise sequences. (a) & (b) SNR = -34.0 dB. (c) & (d) SNR = -34.1 dB. (e) & (f) SNR = -34.2 dB.

sequences. The parameters of the algorithm were set as follows: The TWC size for each PE of the Test Pipe is 3×3×3; the size of the masks in the AP is 3×3. All simulations were run on a Harris HCX9 super minicomputer. To demonstrate the simulation results, Colorado and Lexidata video systems supported by a VAX 11/750 machine were used to display images.

Successful results were obtained with sequences of signal-to-noise ratio well under -30 dB. The signal-to-noise ratio of the sequences was defined as the the decibel value of ratio of the sum of all target pixel energy to the sum of all noise pixel energy in one frame:

$$SNR = 10\log\frac{\sum I_t^2 \mid t \in \{target\ pixels\}}{\sum I_n^2 \mid n \notin \{target\ pixels\}} \qquad \text{(VIII.6)}$$

With Gaussian noise, the signal-to-noise ratio is therefore defined as:

$$SNR = 10\log\frac{\sum \mu_t^2 \mid t \in \{target\ pixels\}}{\sum (\mu_n^2 + \sigma_n^2) \mid n \notin \{target\ pixels\}} \qquad \text{(VIII.7)}$$

where μ_n and σ_n^2 are the mean and variance of the Gaussian noise, respectively. Some examples of both types of noisy sequences and the corresponding detection results are shown below. Figure VIII.5 shows one frame of the pepper-and-salt type noisy sequence and the result of the detection described by the TF representation of the algorithm. One frame of the Gaussian noise sequence and the corresponding TF detection result are shown in Fig. VIII.6. Both examples demonstrate satisfactory detection when the time sequences were mostly obscured by the noise intensities. With lower noise levels (SNR > -34 dB), the system tracks the trajectories with almost no noise effect, while for some very high noise situations (as shown in the examples), the TF contains various noise pixels. However, this should not affect the trajectory type identification. Algorithms like the Hough transform can successfully recognize the trajectories with little effect due to the residual noise.

5. NEURAL NETWORK IMPLEMENTATION OF THE PIPELINE SYSTEM

The SPTD algorithm is based on short term patterns that must be followed by the trajectory. It is well known that artificial neural networks are very well suited for pattern recognition and, consequently, the used of an ANN in the centroid tracker part of the system should improve its performance.

a. BPN Implementation of Centroid Tracker

The centroid tracker can easily and efficiently be implemented by means of a *backpropagation neural network* (BPN) . A BPN is ideally suited to recognize patterns, and since the centroid calculator is essentially a pattern recognizer, the performance of the algorithms should be enhanced by its use.

In the original CT algorithm, PE functions rely on the assumption that consistent distribution of target pixels of a continuous and smooth trajectory will result in a high intensity concentration inside the TWC. So the algorithm does not directly examine the continuity and smoothness. Rather, it measures the condition indirectly by examining the consequence. In fact, the

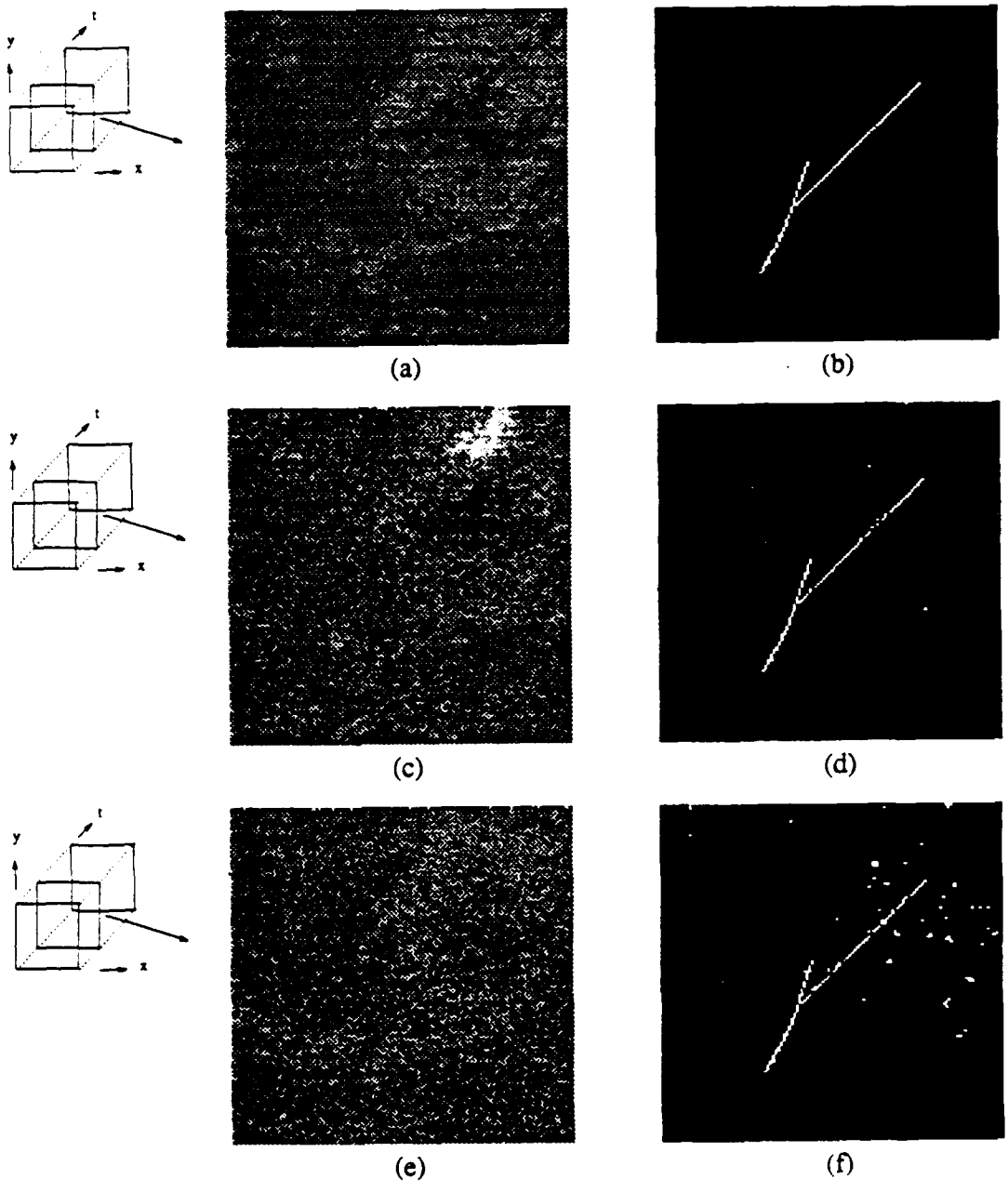


Figure VIII.6 Simulation results by the original system architecture with additive Gaussian noise. (a) & (b) SNR = -33.98 dB. (c) & (d) SNR = -34.08 dB. (e) & (f) SNR = -34.12 dB.

algorithm is invariant to the actual pixel distribution, but only sensitive to the intensity concentration. As the result, more false alarms may be expected, since invalid pixel distributions may also result in a high concentration of intensity in the TWC. For this reason, a pattern recognition scheme is necessary for better PE performance.

Since a BPN implementation incorporates a pattern recognition scheme in its function, a PE would actually seek pixel distributions of the continuous and smooth trajectory patterns. Consequently, it is more discriminating against noise, and PE's implemented by a BPN may conceivably achieve higher noise tolerance, and so experience lower false alarm rates than the original CT algorithm. This constitutes the basic motivation of the implementation. Moreover, upon studying the paradigm, more significant advantages can be found for a BPN implementation: fast response, homogeneous PEs (all equal), hence simplicity of design and training, etc.

b. Test-Pipe Design

With a BPN implementation, each PE is made of a two-layer feed-forward network. Counting the input units as a separate layer, it is, in fact, a three-layer network with an input layer, a hidden layer and an output layer, as illustrated in Fig. VIII.7 Consequently, the TP consists of an $N \times N$ array of PE networks of homogeneous structure and functionality. The network input is taken from the PE's TWC. Hence, it is of dimension $D \times w \times w$.

The input layer consists of all the data nodes connected to the PE which are within the TWC. So there are $D \times w \times w$ input units in the layer, 27 for the 3×3×3 TWC. Each input unit is connected to a particular pixel in the TWC in a sequential order. The input layer is reloaded in every cycle when the TP is updated with a new image frame. Each input signal is a continuous value within [0, 1] by normalizing the input grey-level intensity (between 0 and 255) by 255.

The output layer consists of one unit that produces positive signals between [0, 1]. It corresponds to the single output of the PE. It is then thresholded to be either "high" or "low" which sets or resets the corresponding pixel of the TFB indicating the position to be a positive centroid point or otherwise. The threshold value depends on the convergence of the network and is determined by the training process. In general, the signal value of the output unit has a large separation between positive and negative responses when the network converges. Hence, the threshold value is fairly easy to choose.

A hidden layer with $2I+1$ units has been adopted for the PE structure, I being the number of input units. Two connection geometries have been considered for the PE structure: a fully connected network and a network with receptive field connection. The fully connected network is as described before, one in which each unit is connected to every unit in the layer immediately below it. In this geometry, the activation function of a unit is simply a weighted sum of all the output signals of the lower layer, given by

$$a_j = \sum_{i=1}^{N} w_{ji} o_i \qquad \begin{cases} 1 \le j \le 55 & \text{for } \textit{hidden layer} \\ j = 1 & \text{for } \textit{output layer} \end{cases} \qquad \text{(VIII:8)}$$

where N = 27 for the input layer, and N = 55 for the hidden layer; o_i is the output signal of the input or hidden units with layer j being the hidden or output layer.

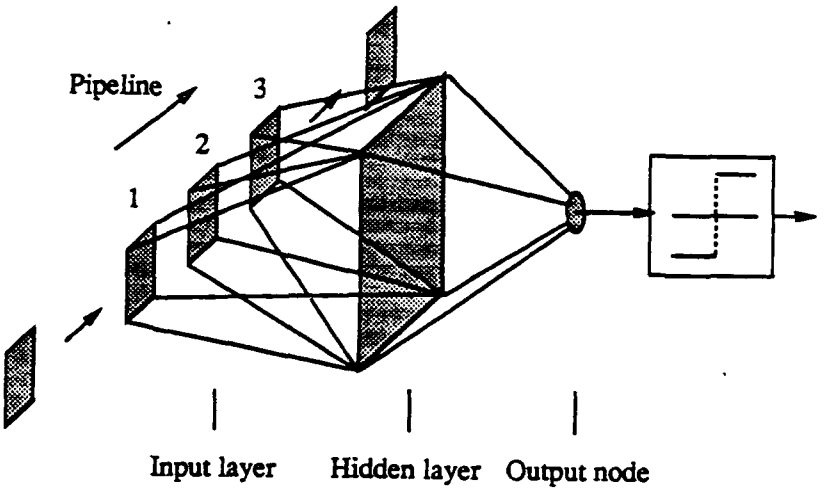


Figure VIII.7 BPN-implemented PE structure

c. Network Training

The learning procedure for the PE network is based on the generalized delta rule, as described in section VIII.5.a. It basically involves presenting a set of input and output vector (pattern) pairs to the network. After forward propagating an input vector to the output, the system computes an error vector (a scalar in the PE network due to the single output line) and propagates it backward to the hidden layer according to which the weight values are updated. This section presents details of the learning process in terms of the weight adaptation, training set, training procedure, and some important training issues.

(c.1) Weight Adaptation

The weight adaptation rule chosen includes a momentum term. Let o_i, o_j and o_k denote the output signal of an input unit, hidden unit and output unit, respectively, where $i \in \Psi$, $1 \le j \le 39$ and $k = 1$. Then, for the weights connected to the output unit,

$$\Delta w_{kj}(t) = \eta \delta_k o_j + \alpha \Delta w_{kj}(t-1) \qquad k = 1,\ 1 \le j \le 39 \tag{VIII.9}$$

The error signal δ_k is given by

$$\delta_k = o_k(1 - o_k)(d_k - o_k) \tag{VIII.10}$$

where d_k is a desired output signal, and $d_k = \{0, 1\}$.

Between the hidden layer and the input layer,

$$\Delta w_{ji}(t) = \eta \delta_j o_i + \alpha \Delta w_{ji}(t-1) \qquad 1 \le j \le 39,\ i \in \Psi \tag{VIII.11}$$

where the error signal δ_j is given by

$$\delta_j = o_j(1 - o_j)\delta_k w_{kj} \qquad k = 1 \tag{VIII.12}$$

In practice, α is generally chosen to be greater than η for faster convergence. In the actual training sessions, the learning rate $\eta = 0.4$, and the momentum gain $\alpha = 0.6$ have been used.

(c.2) Training Set

The training set contains patterns of 27 digits, corresponding to 27 pixels in the TWC. Each pattern thus consists of three 3×3 squares of integer numbers representing grey-levels in digital image, each of the squares corresponding to one of the windows. The pixel intensity is of binarized values of {0, 200}, indicating either low or high intensity level. The high intensity is selected as 200 out of maximum intensity value of 255 based on the mean target intensity used in the simulations. The desired output is of binary values {0, 1}. "1" indicates that the output unit is on and represents a positive response associated with input patterns in the positive class. On the other hand, "0" indicates the output unit is off and represents a negative output response associated with negative input class patterns.

The selection of training patterns is very much task dependent. For different kinds of target trajectories, different training sets should be employed. The reason is that adequate information about features of different target trajectories should be emphasized in selecting positive training patterns. The analogy is true, too, for the negative training set where different features of noise distributions need to be emphasized. Therefore, it is highly recommended that the user of this

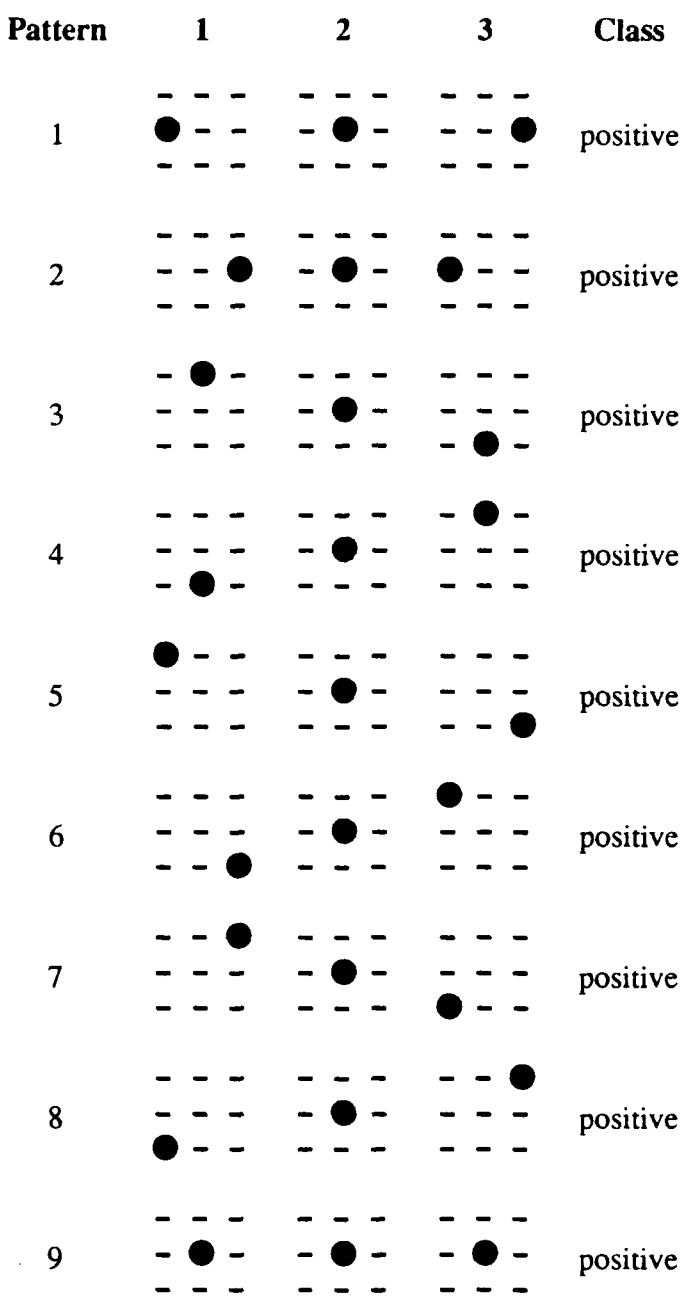


Figure VIII.8 Straight-line pattern vectors, ●: high intensity, - : low intensity.

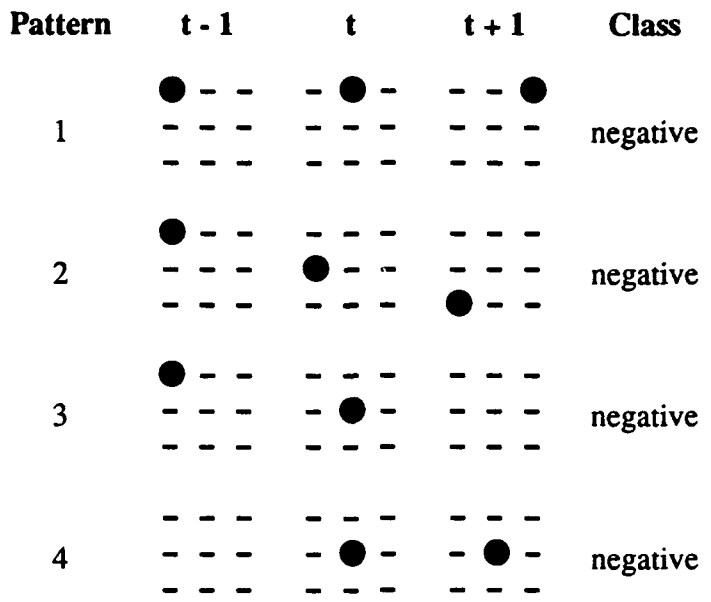


Figure VIII.9 Examples of negative class patterns, ●: "on" pixel, - : "off" pixel.

network study the possible target images, and create different training sets for different target situations for better performance of this system.

Figures VIII.8, 9 and 10 shwo examples of training patterns, some correponding to valid trajectories (*positive*) and some to invalid ones (*negative*.)

Figure VIII.11 shows simulation results for a salt-pepper type of noise using the BPN. Compare these results with those in Fig.VIII.5 .

d. Hardware Implementation Issues

In this section, the hardware requirements for the BPN implementation of the TP-PE's are considered. With the BPN implementation, each TP-PE consists of a BPN structure, instead of the ALU-based digital processing element described in the previous section. As was discussed in Section VIII.5.a, the BPN is basically made up of a network of connection weights and weighted sum and sigmoid units. When implemented in hardware, these function units may be realized by specially designed analog circuits.

The input layer is composed of 19 units, each connecting to the respective data nodes in the receptive field of the 3×3×3 TWC space. Each input unit is only a data buffer containing a normalized pixel value that can be implemented by a fixed-length register. So, the input layer contains a register space of 19 such registers.

The hidden unit (or the output unit) basically consists of a summing circuit which implements the summing operation and a sigmoid circuit that realizes the sigmoid function. The only

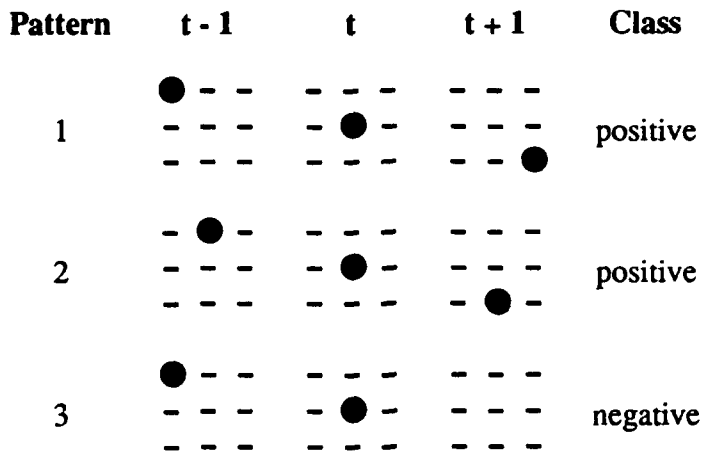


Figure VIII.10 Training pattern examples

difference between a hidden unit and an output unit is the number of the input signals to the summing circuit (implemented with summing amplifiers), 19 for the hidden and 39 for the output unit. There are 39 hidden units and one output unit in the network. Therefore, each PE consists of 40 summing circuits and 40 sigmoid circuits.

A connection weight of the BPN can be implemented by a resistor. In the network, there exist 741 (19×39) weights between the input and hidden layers, and 39 (39×1) weights between the hidden and output layers. Thus, there are a total of 780 weight resistors in the network.

In summary, a PE network consists of 40 sigmoid circuits, 40 summing circuits, 19 registers (data buffers), and 780 weight resistors. At the TP array processor level, each number will be multiplied by the number of PE's in the array due to the homogeneous PE structure. Table VIII.1 summarizes the results in terms of the general $N \times N$ PE structure of the TP corresponding to $N \times N$ image frames. If, for example, 128×128 image frames are considered, there will be approximately 655,000 sigmoid circuits, 655,000 summing circuits, 311,000 registers, and 12,700,000 weight resistors required for the TP array processor. These are very large numbers. However, when several PE's are integrated into one VLSI chip and the TP integrated at the board level, the system is believed to be implementable with current technology.

Table VIII.1 Hardware requirements for a $N \times N$ TP

Sigmoid Circuits	Summing Circuits	Registers	weight resistors
$40N^2$	$40N^2$	$19N^2$	$780N^2$

6. CONCLUSIONS

This final report has described the design and development of a pipeline system, i.e., the Pipeline Target Detection System, for the real-time task of single-pixel target detection and tracking. An original method has been designed for the problem from the task specifications of a space-based ICBM-interception problem, based on which algorithms and consequently a system implementation have been developed.

SECTION IX

HOUGH TRANSFORM SINGLE PIXEL-TARGET DETECTION

1. INTRODUCTION

Once a target trajectory image is obtained from the *Single pixel target trajectory detection system*, it is necessary to identify valid targets and to determine the coordinates of the target in the last frame. This is done by means of a system based on the Hough transform. We have developed an original mapping scheme that allows analog parallel implementation of the Hough transform, as well as an ANN implementation of the peak detector.

2. PARALLEL MAPPING SCHEME

The parameter plane used in this mapping scheme is based on the parameterization proposed by Jain et al.[1] in which three sides of the image plane are used for the so called circumference parameterization. The two parameters used are: slope and line intersection with the upper, right and lower sides of the image.

This parameterization has the advantages of being uniform in quantization errors and to have a bounded range of values of both parameters θ and η.

a. Determination of η Values

According to the definition, in the theta-circumference parameterization, we have to find values of η for $0° \leq \theta \leq 180°$. The angle θ varies form 0° to 180°. In this way the three sides of the image plane are covered. Formulas for ranges of θ from 0 to 90° and 90 to 180° have been developed in which the value of η is also taken into account.

b. Mapping Scheme

The mapping scheme allows the transformation from image plane to parameter plane to be done completely in parallel.

Figure IX.1. shows the mapping structure. The image plane can be viewed as many "θ-layers." Each "layer" corresponds to a value of θ (discrete value of slope). A "layer" is a mapping of the image plane to a cell of parameter plane in which every pixel of image plane is assigned a value of η in parameter plane. The figure also shows three lines in the image plane, from three θ-layers: θ=45°, θ=90°, θ=135°. A solid line indicates a line with slope equal tothe θ value of the layer. All pixels in the image plane that belong to a line are connected, using a summer, to a cell in parameter plane with the (η_i, θ_i) value of that line.

If a line with this (η_i, θ_i) appears in the image, for example (η=2, θ=45°), the (η_i, θ_i) cell in the parameter plane will have (in this low resolution example) a count of *eight*. For other lines like $(\eta_i$, (η=13 , θ=90°) and (η=6, θ=135°), the count produced in the same (η_i, θ_i), at (η=2, θ=45°) cell, will be very low (one, in this example). In addition, the figure shows that for θ=90° and θ=135° "layers," the line at (η=2, θ=45°) contributes a count of one to the (η=13, θ=90°) and (η=6, θ=135°) cells.

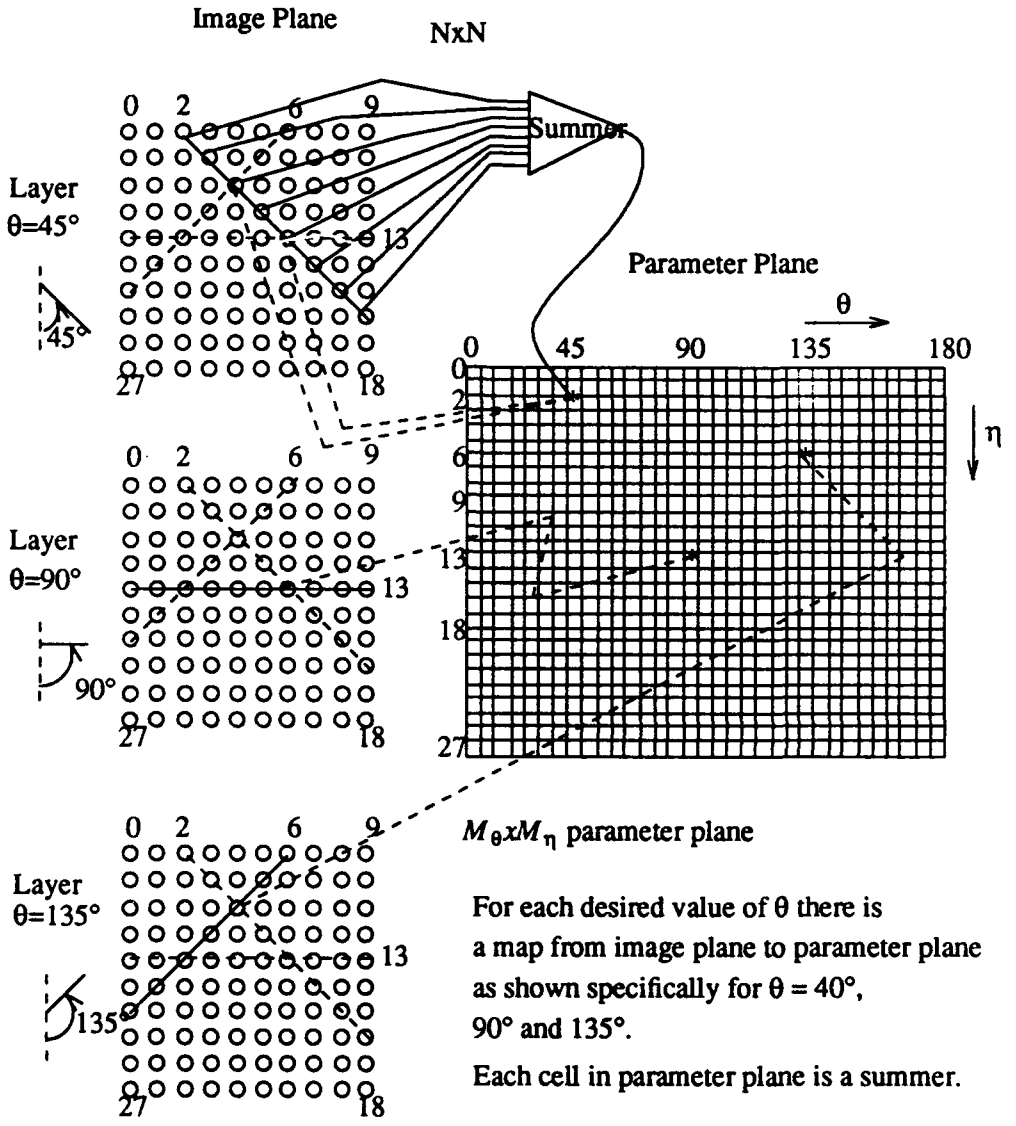


Figure XI.1. Mapping Scheme.

c. Connection and Amplifier Requirements for Mapping Scheme

To implement this architecture with analog circuitry, every image point should be connected to an amplifier corresponding to each value of η in each θ "layer". If the resolution of the image plane is NxN and the resolution of parameter plane is $M_\theta x M_\eta$, theoretically we need $NxNxM_\theta$ connections if all pixels in the M_θ "layers" were used. Each node in parameter plane is implemented by means of an operational amplifier summer. In total, $M_\theta x M_\eta$ amplifiers would be needed for parameter plane.

3. PEAK DETECTION

The next step in the line detection process involves identifying the peaks and their location in the parameter plane. The time needed for detecting peaks should be comparable to the time needed for transforming the image plane to the parameter plane. Since the latter is done in a parallel manner, a parallel solution to the peak detection process is needed if the two times are to be comparable.

Use of a straightforward peak detection scheme with the existence of spurious peaks would result in the detection of non-existent lines. Thus there is a need to eliminate the spurious peaks from the parameter plane. This elimination is done by a combination of (a) Thresholding, (b) Main windows, (c) Overlapping windows. The scheme proposed for peak detection is shown in Fig.IX.2.

Thresholding eliminates peaks that have values less than some specified threshold. The threshold value is generally equal to the minimum line length that is to be detected and is application dependent. Typically, thresholding eliminates a majority of the spurious peaks.

a. Main Windows

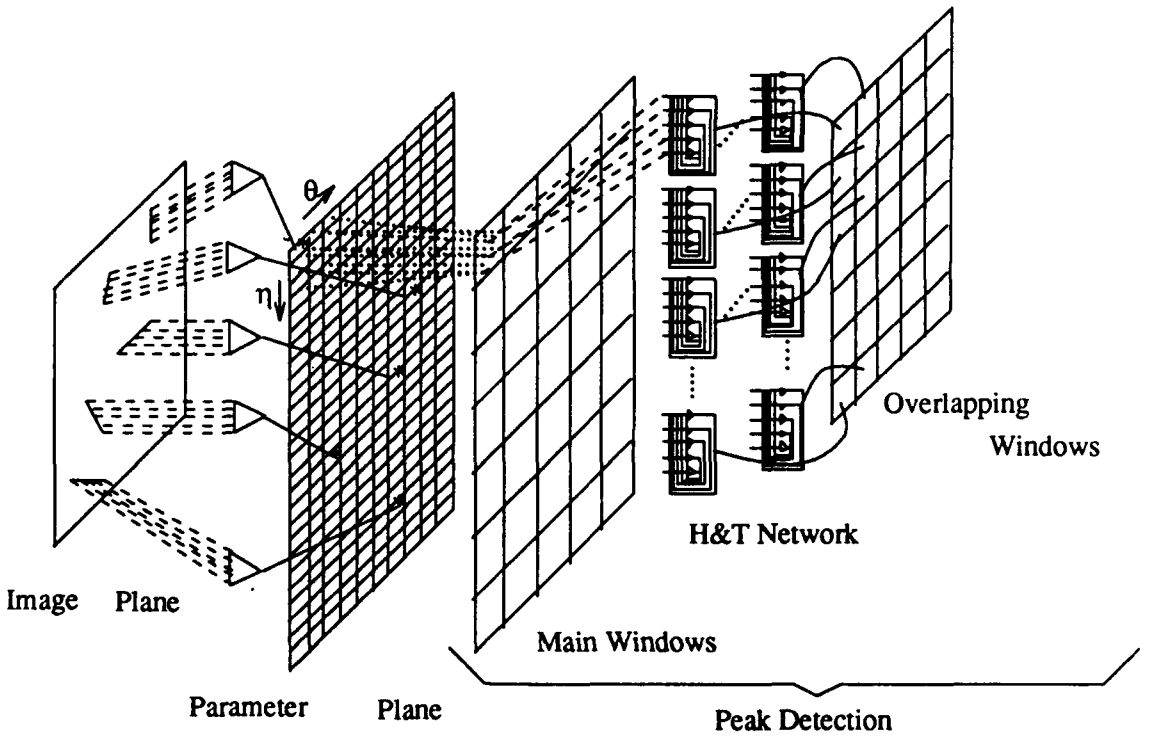


Figure IX.2. Line Detection Scheme.

When input images have lines of varying length, the threshold value is set to the minimum line length. However, the HT process produces spurious peaks that may have values close to the actual peak and greater than the threshold. Hence, these peaks remain after thresholding. The actual peaks may be distributed across the parameter plane and may have widely differing values.

A peak detection scheme that uses the entire parameter plane as its single input would result in only the longest lines being detected. Thus, it is necessary to divide the parameter plane into subsections called *windows*, and apply the peak detection process within each of these subsections. Windows of size nxn start at $\theta=0°$ and $\eta=0$ and do not overlap.

For the parameter plane of size 382x60, there are 900 windows of size 5x5 and 12 windows of size 7x5. However as explained previously, certain areas in the parameter plane always have zero count and no windows are needed in this areas. This decreases the number of windows needed to 584 of size 5x5 and 7 of size 7x5.

b. Hopfield and Tank Network Approach for Peak Detection

Peak detection within each main window can be done in a parallel manner using an artificial neural network. The network must be capable of detecting peaks of arbitrary value and provide as output the value and location of the peak. The Hopfield-Tank network provides an efficient (both in time and space) implementation of such a peak detector.

A Hopfield-Tank peak detector used in the calculation of centroids of images has been developed. This peak detector is useful for our application an will be used in this system also. The structure of this network and its *energy function* are the same as proposed by Hopfield-Tank [2]. The neuron however uses a one-sided shifted sigmoid function, see Section IV of report *E. Susbsystem Report-Algorithms* .

c. Overlapping Windows

As described in the previous sections, the parameter plane is divided into main windows that have fixed positions. Peak detection using the Hopfield-Tank network is done on each of these main windows. However, it is possible that a peak corresponding to a line has its neighborhood area spread over into the adjacent windows where the spurious peaks may be detected as lines.

In order to overcome this problem, a scheme of overlapping windows is used. While the main windows have fixed positions, an overlapping window is centered about a peak detected in a main window. Consequently, overlapping windows have variable position and should be implemented in software. This will not represent a considerable increase in processing time, because the number of overlapping windows is small (typically less than ten).

Using a 5x5 overlap window centered about an actual peak, produced the best results. In order to treat all peaks similarly, an overlapping window on the left or right edge of the parameter plane is wrapped around onto the opposite edge.

The combination of analog hardware implementation and software overlapping windows could be implemented as follows: the parameter plane can be shifted out of the chip, for example row by row, into some disk file or directly into memory under program control. The peak location information is obtained from the Hopfield-Tank network. At this stage the overlapping window can be applied in software.

4. TARGET LOCATION

The output of the *Single Pixel Target Detector* (SPDT) is an image containing possible trajectories and noise pixels. The *Hough Transform* (HT) extracts valid straight line trajectories from this image.

It remains to determine what point (i.e., pixel) corresponds to the actual target (or targets for more than one valid trajectory.) This information is not directly provided by either of the two algorithms.

The SPTD, however, works with a sequence of frames and pixels kept in the last frame and their coordinates are available.

From the HT output we know the equations of the detected straight lines trajectories. If we replace all the pixels selected in the SPTD last frame in all these equations, only the equations to which the pixels actually belong will produce the same "η" value than the trajectory detected in the HT (using the formulas developed in section.2). This, then, determines the target-pixel location at the instant of time corresponding to frame "n", the last frame in the sequence. This procedure is implemented in the SPTD program.

5. VLSI IMPLEMENTATION ANALYSIS

This section provides a brief analysis of the hardware needed for the parallel HT implementation presented previously. The hardware required for implementation of the parallel HT is mostly composed of operational amplifiers (op-amps) and resistors.

a. Processing Elements for the Mapping Scheme

Operational amplifiers are used in two phases of the system: mapping from image plane to parameter plane and the Hopfield-Tank peak detectors. The op-amp configuration used is shown in Fig. IX.3. From the figure, and ignoring the diode for the moment, we have:

$$V_o = V_A + R_f\left[\frac{V_A}{R_1}+\frac{V_A}{R_2}+\cdots+\frac{V_A}{R_n}\right] - R_f\left[\frac{V_1}{R_1}+\frac{V_2}{R_2}+\cdots+\frac{V_n}{R_n}\right]$$

where,

$$V_A=\frac{R_B}{R_A+R_B}V_T$$

Select

$$\mathrm{R} = R_1 = R_2 = \ldots\ldots = R_n = R_f$$

then

$$V_o = V_A(1+N)-(V_1+V_2+\ldots+V_n)$$

Now considering the diode, when the sum of the input voltages is greater than $V_A(1+N)$, the diode remains cut-off and the desired output, V_o, is available. When $V_A(1+N) > (V_1+V_2+\ldots+V_n)$, the diode is turned-on and $V_o \approx 0.8$. The op-amp is short-circuit protected and the current through the diode is limited. However, the threshold value $V_A(1+N)$ depends on N, the number of connections to the summer. This number may be different for different (η,θ) pairs. In order to provide a fixed threshold for all the summers, the voltage V_A is derived from the resistor pair R_A and R_B with the R_A value being chosen such that the product

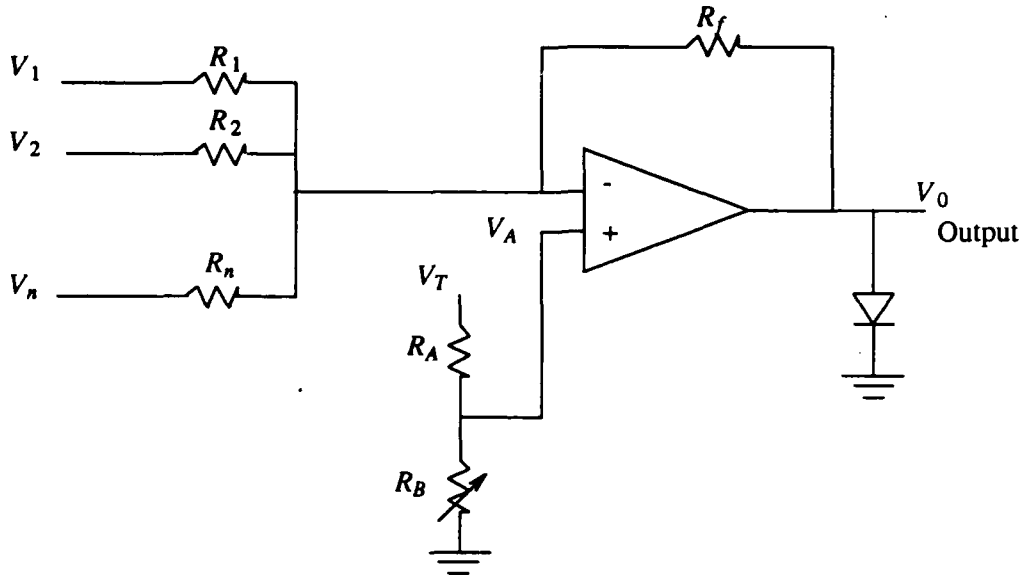


Figure IX.3. Parameter Plane Summer with Thresholding.

$V_A(1+N)$ remains constant for different values of N. This scheme provides the same threshold for all summers.

b. Connections, Amplifiers and Feasibility Study

The parameter plane mapping would require a maximum of $M_\theta x M_\eta$ processing elements. However, as explained in section 2, certain (η,θ) combinations will always have zero count and no summers need be connected to these cells of the parameter plane.

For this application, N = 128, $M_\theta = 60$ (three degree theta increments), $M_\eta = 382$.

The experiment described in section 2, showed that 7,818 cells of the parameter space have zero count. Thus, the actual number of summers needed is:

$$(382x60)-7,818 = 15,102.$$

The window system generates 912 windows of which 900 have a size of 5x5 and 12 have a size of 7x5 cells of the parameter space. A Hopfield-Tank network is used for peak detection in each of the above windows. However, as explained before, many of the parameter plane cells have fixed zero counts and need not be connected to the peak detection system. Hence, the peak detection system needs 15,102 neurons. The neuron is the same than one used in Centroid Calculation, and uses a single op-amp.

Therefore, the total number of op-amps needed by the two phases is:

Number of op-amps in mapping scheme ≈ 15,102
Number of op-amps in peak detector ≈ 15,102

Total ≈ 30,204 op-amps

The number of weights (resistors) needed is much higher. For the mapping scheme, it is about a million and for the peak detector about 400,000. This means that a VLSI implementation is not possible yet, because current technology allows a chip area of 4 cm^2 and the area needed would be 10 cm^2. A multiple chip implementation is certainly possible. The mapping and peak detector schemes may be distributed across the minimum number of chips needed to accommodate the hardware. The calculations done previously indicate that very few chips would be needed.

Another possibility is the use of wafer-scale integration. The process is prone to fabrication defects, in other words the yield is low. This problem may be solved by using redundancy. If the wafer contains "n" circuits, only k < n will actually be needed.

6. SIMULATIONS

The algorithm has been extensively simulated with both synthetic images and with trajectory images provided bu the SPTD system. A full report is given in *C. Software test-Simulations*. In this report we will limit ourselves to a couple of significant example. Figure IX.4 shows an image with five synthetic trajectories. Notice the crossing of trajectories and the "bad" location of the vertical trajectory near the edge of the image. Table IX.1 shows the result of the Hough transform for this image. Tables IX.2(a),(b),(c) and (d) show, respectively, the accumulator result, the accumulator after thresholding, after main window algorithm and after overlapping windows for the horizontal (i.e., 90 δ) line. The final result is a peak at θ=90 δ and η=156, i.e., the correct result. The results for the other lines are also correct, although their windows are not shown.

Figure IX.5 is a noisy input image with a straight line trajectory and a curved one. This image is generated by the SPTD system. The Hough algorithm, as implemented here, only recognizes straight lines. The interesting fact is that the curve has been recognized as three straight line segments, as the result in Table IX.4(c) shows. The peak at θ=135 δ and η = 127 corresponding to the straight line, is correctly detected but is not shown in the tables.

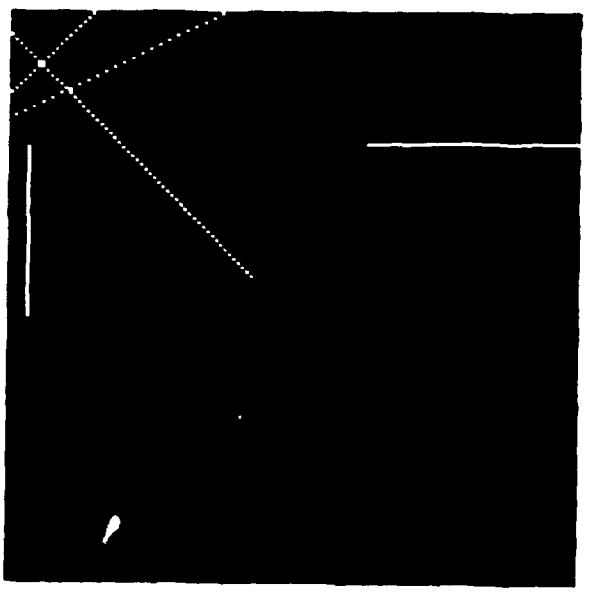


Image GINP0. Five different lines

Figure IX.4 Synthetic image with five lines

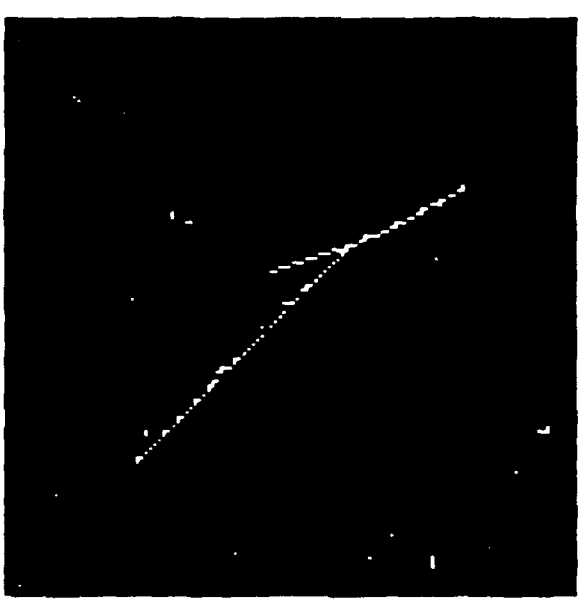


Figure IX.5 Noisy trajectory output of SPDT system

Table IX.1 SIMULATION RESULTS OF HOUGH TRANSFORM

Line	Theta (θ)	Eta (η)	Peak (points)
θ=0°, η=4, peak=39	**0°**	**4**	**41**
	3°	1,2	21,21
θ=45°, η=259, peak=55	**45°**	**259**	**55**
θ=90°, η=156, peak=48	87°	157,158	21,20
	90°	**156**	**49**
	93°	154,155	21,21
θ=117°, η=48, peak=24 (*Actual line at θ=116.57°*)	**117°**	*46*,**47**	*10*,**14**
θ=135°, η=18, peak=19	**135°**	**18**	**19**
	138°	17	10
θ=179°, η=5, peak=42 (*This is not a true line, it is due to the line at θ=0°*)	177°	6,7	**21,21**

Simulation Parameters:
Input image GINP0.
Five different lines.

Table IX.2 SIMULATION RESULTS OF PEAK DETECTION SCHEME

η \ θ	75	78	81	84	87	90	93	96	99	102
151	0	0	1	3	3	1	2	7	8	6
152	0	0	3	3	3	1	2	11	8	7
153	0	0	2	3	2	1	2	12	8	6
154	0	0	3	3	2	1	21	11	8	6
155	0	2	4	3	2	1	21	12	8	7
156	0	4	6	8	11	49	12	7	6	5
157	0	7	9	13	21	2	2	2	2	2
158	0	5	9	12	20	2	2	2	2	2
159	0	7	10	11	1	2	2	1	2	1
160	0	8	8	11	2	2	2	2	1	2
161	3	8	9	8	1	2	2	2	2	2
162	5	7	10	2	1	2	2	2	2	2
163	4	7	9	2	2	2	2	2	2	2
164	6	7	2	1	2	2	2	2	2	2
165	6	9	3	1	2	2	2	2	2	1
166	6	5	3	1	2	2	2	2	2	2

Table 5.1.a Simulation Results GINP0.
(θ = 90°, η = 156, peak = 48).
Parameter Plane Representation.
Three Degree θ Increments.

η \ θ	75	78	81	84	87	90	93	96	99	102
151	0	0	0	0	0	0	0	0	0	0
152	0	0	0	0	0	0	0	0	0	0
153	0	0	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	21	0	0	0
155	0	0	0	0	0	0	21	0	0	0
156	0	0	0	0	0	49	0	0	0	0
157	0	0	0	13	21	0	0	0	0	0
158	0	0	0	0	20	0	0	0	0	0
159	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0
161	0	0	0	0	0	0	0	0	0	0
162	0	0	0	0	0	0	0	0	0	0
163	0	0	0	0	0	0	0	0	0	0
164	0	0	0	0	0	0	0	0	0	0
165	0	0	0	0	0	0	0	0	0	0
166	0	0	0	0	0	0	0	0	0	0

Table 5.1.b Simulation Results GINP0 .
(θ = 90°, η = 156, peak = 48).
After Thresholding (threshold = 12).

η \ θ	75	78	81	84	87	90	93	96	99	102
151	0	0	0	0	0	0	0	0	0	0
152	0	0	0	0	0	0	0	0	0	0
153	0	0	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	0	0	0	0
155	0	0	0	0	0	0	0	0	0	0
156	0	0	0	0	0	49	0	0	0	0
157	0	0	0	0	0	0	0	0	0	0
158	0	0	0	0	0	0	0	0	0	0
159	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0
161	0	0	0	0	0	0	0	0	0	0
162	0	0	0	0	0	0	0	0	0	0
163	0	0	0	0	0	0	0	0	0	0
164	0	0	0	0	0	0	0	0	0	0
165	0	0	0	0	0	0	0	0	0	0
166	0	0	0	0	0	0	0	0	0	0

Table 5.1.c Simulation Results GINP0.
(θ = 90°, η = 156, peak = 48).
After Main Windows.

η \ θ	75	78	81	84	87	90	93	96	99	102
151	0	0	0	0	0	0	0	0	0	0
152	0	0	0	0	0	0	0	0	0	0
153	0	0	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	0	0	0	0
155	0	0	0	0	0	0	0	0	0	0
156	0	0	0	0	0	49	0	0	0	0
157	0	0	0	0	0	0	0	0	0	0
158	0	0	0	0	0	0	0	0	0	0
159	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0
161	0	0	0	0	0	0	0	0	0	0
162	0	0	0	0	0	0	0	0	0	0
163	0	0	0	0	0	0	0	0	0	0
164	0	0	0	0	0	0	0	0	0	0
165	0	0	0	0	0	0	0	0	0	0
166	0	0	0	0	0	0	0	0	0	0

Table 5.1.d Simulation Results GINP0.
(θ = 90°, η = 156, peak = 48).
After Overlapping Windows.

Table IX.3 SIMULATION RESULTS OF PEAK DETECTION SCHEME

η \ θ	93	96	99	102	105	108	111	114	117	120	123	126	129	132	135	138	141
124	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2
125	0	0	0	0	0	0	0	0	0	0	0	0	0	1	5	1	1
126	0	0	0	0	0	0	0	0	0	0	0	0	0	2	14	2	1
127	0	0	1	0	0	0	0	0	0	0	0	0	0	1	44	2	3
128	0	0	0	0	0	0	0	0	2	0	0	0	1	3	4	1	3
129	0	0	0	0	0	0	0	0	0	0	0	0	3	3	2	2	0
130	0	0	0	0	0	0	0	1	0	0	0	0	2	1	1	2	2
131	0	0	0	0	0	0	0	1	0	0	0	0	2	3	2	3	3
132	0	1	0	0	0	0	0	0	0	1	0	0	2	11	3	3	0
133	0	1	0	0	0	0	0	1	0	0	0	0	2	14	2	0	3
134	0	0	0	0	0	0	0	1	0	0	0	3	3	18	2	2	1
135	0	0	0	0	0	0	1	0	0	0	0	2	1	12	3	3	0
136	0	0	0	0	0	0	1	0	0	0	0	3	2	14	3	3	0
137	0	0	0	0	0	0	0	0	0	0	0	2	11	5	2	1	0
138	1	0	0	0	0	0	0	0	0	0	0	3	11	4	3	0	0
139	1	0	0	0	0	0	2	0	1	0	3	3	6	4	3	0	0
140	0	0	0	0	0	0	0	0	0	0	3	2	12	3	1	0	0
141	0	0	0	0	0	1	0	0	0	0	3	7	14	4	0	0	0
142	0	0	0	0	0	1	0	0	0	0	3	7	10	3	0	0	1
143	0	0	0	0	0	0	0	0	0	0	3	11	10	0	0	0	0
144	0	0	0	0	0	2	0	0	0	4	4	9	11	0	0	0	0
145	0	0	0	0	0	0	0	1	0	4	7	10	7	0	0	0	0
146	0	0	0	0	1	0	0	0	0	3	8	9	1	0	0	1	0
147	0	0	0	0	1	0	0	0	0	5	13	8	0	0	0	0	0
148	0	0	0	0	0	0	0	0	2	6	14	8	0	0	0	0	0
149	0	0	0	0	2	0	0	0	6	11	4	5	0	0	1	0	0
150	0	0	0	0	0	0	0	0	6	19	4	4	0	0	0	0	0
151	0	0	0	1	0	0	1	0	5	5	1	5	0	0	0	0	1
152	0	0	0	1	0	0	0	0	24	5	3	4	1	1	0	0	0
153	0	0	0	0	0	0	0	9	10	2	7	4	0	0	0	0	0
154	0	0	0	2	0	0	1	21	2	2	4	2	0	0	0	0	0
155	0	0	0	0	0	0	4	20	3	3	3	0	1	0	0	0	0
156	0	0	1	0	0	1	7	2	3	1	5	0	0	0	0	0	0
157	0	0	1	0	1	6	23	1	3	3	5	1	0	0	0	0	0
158	0	0	2	0	3	6	15	2	1	4	4	0	0	0	0	0	0
159	0	0	0	2	4	6	1	3	3	5	3	0	0	0	0	0	0
160	0	0	1	3	4	12	2	3	2	2	5	0	0	0	0	0	0
161	0	2	2	4	5	20	1	2	1	3	3	0	0	0	0	1	0
162	0	3	4	3	7	1	2	2	2	4	0	1	0	0	0	0	0
163	1	4	3	5	7	2	3	1	3	4	0	0	0	0	0	0	0
164	2	2	3	4	15	1	2	3	6	4	0	0	0	0	0	0	0
165	5	4	3	5	6	2	3	1	2	5	0	0	0	0	0	0	0
166	4	3	3	5	1	1	1	3	3	2	0	0	0	0	0	0	0
167	4	2	4	10	2	3	1	2	2	3	0	0	0	0	0	0	0
168	2	3	5	9	1	1	4	2	3	3	0	0	0	0	0	0	0
169	2	3	5	3	2	3	1	5	3	0	0	0	0	0	1	0	0
170	3	5	7	2	2	2	1	2	4	0	0	0	0	0	0	0	0
171	2	2	6	1	2	1	2	3	3	0	1	0	0	0	0	0	0

Simulation Results Noisy Input Image. Target Seeker Trajectory.
($\theta = 135°$, $\eta = 127$, peak = 44) & Representation of a curve.
Parameter Plane Representation (Three Degrees θ Increments).

Table IX.4 SIMULATION RESULTS OF PEAK DETECTION SCHEME

(a)

η \ θ	90	93	96	99	102	105	108	111	114	117	120	123	126	129	132	135	138	141
147	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
148	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
149	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
151	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
152	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0
153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	0	0	21	0	0	0	0	0	0	0	0	0
155	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0
156	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
157	0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0	0	0
158	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
159	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
161	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
162	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
163	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
164	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Simulation Results Noisy Input Image. Target Seeker Trajectory.
(θ = 135°, η = 127, peak = 44) & Representation of a curve.
After Thresholding (threshold = 19).

(b)

η \ θ	90	93	96	99	102	105	108	111	114	117	120	123	126	129	132	135	138	141
147	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
148	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
149	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
151	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
152	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0
153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
155	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
156	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
157	0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0	0	0
158	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
159	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
161	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
162	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
163	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
164	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
165	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Simulation Results Noisy Input Image. Target Seeker Trajectory.
(θ = 135°, η = 127, peak = 44) & Representation of a curve.
After Main Windows.

Table IX.4 SIMULATION RESULTS OF PEAK DETECTION SCHEME

(c)

η \ θ	90	93	96	99	102	105	108	111	114	117	120	123	126	129	132	135	138	141
147	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
148	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
149	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
151	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
152	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0
153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
155	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
156	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
157	0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0	0	0
158	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
159	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
161	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
162	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
163	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
164	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
165	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Simulation Results Noisy Input Image. Target Seeker Trajectory.
($\theta = 135°$, $\eta = 127$, peak = 44) & Representation of a curve.
After Overlapping Windows (threshold = 19).

SECTION X

CONCLUSIONS AND RECOMMENDATIONS

1. CONCLUSIONS

The goal of this research was to develop an integrated target seeker system capable of recognizing a target in a scene where other objects, such as decoys could be present. The problem was divided in two parts: multipixel target and single pixel target.

For the multipixel target, we first investigated the use of the M-transform (a translation invariant transform,) and the neocognitron. After applying the M-transform to the image in the original rectangular tessellation image, invariance to image translation would be achieved. If the M-transform was applied to the computation plane image of the LSM, invariance to rotation and tranlations would be achieved. The neocognitron would then be used for recognition, in either case.

During the course of the research, it was discovered that this approach presented several problems, as follows:

a. Translations, rotations and scaling could not be handled simultaneously

b. The M-transform is **not** one-to-one (this was shown by developing different synthetic patterns all of which mapped to the same M-transformed pattern). Consequently, this is not a robust method of pattern identification and classification.

c. The neocognitron itself is not a reliable classifier for complicated and noisy patterns. This was also shown by comparing the results of neocognitron classification with those of a first and a second order backpropagation network. The BPNs provided much better results.

It was decided, consequently, to use a different approach which is the one described in this report.

After producing a binary edge image by means of the motion/edge detection algorithm, and segmenting the image to windows containing individual objects by means of the segmentation algorithm, the centroid of the gray level image was determined by means of the parallel analog network/Hopfield-peak-detector combination. Notice that, after image segmentation and windowing, all subsequent subsystems can operate on each window simultaneously (in parallel).

The original motion/edge detection algorithm was significantly modified in such a way that the number of required operations was reduced by orders of magnitude and it is now possible to implement the algorithm in real time.

We feel that the design of the analog connectionist network centroid calculator represents a significant contribution, not only because it is an original design, but because of the possibility of implementing it in VLSI. The SPICE simulations produced excellent agreement with the theory and we showed that VLSI implementation is possible using current technology.

The centroid subsystem serves a dual purpose: It keeps the object's image centered on the centroid, thus allowing simpler recognition by the MHONN, and also serves as a tracker of object translation, by computing the displacement of the centroid from frame to frame. Notice

that scaling or rotations on the optical axis do not produce centroid displacements.T)

After the centroid of the object in each window is determined, the window is centered on the centroid and then log-spiral-mapped (using arcs-of-ring tessellation) to the computation plane. Two hardware approaches to the rectangular-to-exponential tessellation mapping have been proposed, one hybrid and the other digital. Both are easily implementable using special purpose circuitry.

The modified higher order neural network classifies the object in each window. Due to the use of the centroid calculator, this network has to deal only with scalings and rotations, which, due to the use of the LSM computation plane image, become equivalent to translations on the vertical and horizontal axes, respectively. Simulations have shown that the MHONN is very robust and produces reliable results not only for objects of different sizes and orientations, but for images with significant noise content.

The MHONN was significantly improved by means of an architectural change that allows a reduction of orders of magnitude in the number of afferent weights. This, in turn, will make it possible to implement the proposed hybrid circuit using VLSI technology. As is the case with the centroid calculator, we believe that both the new HONN design and the proposed hardware implementation represent significant contributions.

The line-correlator-target-tracker has the ability of tracking displacements either in the rectangular tessellation image plane, or in exponential tessellation computation plane. By using it in the latter, rotations and scalings can be determined. Combining the centroid tarcking of displacements with the LCT tracking of rotation/scaling, an accurate account of changes in x-y-z position of the object in the window selected by the MHONN, as well as changes in orientation, is achieved. A Hopfield-Tank neural network implementation of the LCT has been designed and can be implemented in VLSI.

The above summary of the multipixel branch of the system, shows that the goals of the project have been achieved. What remains to be done is to design the hardware implementation of some parts of the system, namely the motion/edge detector and segmentation/windowing.

The single pixel target system consists of two subsystems: the target trajectory detection system and the Hough transform line detection system. A single pixel target can be detected only based on trajectory continueity. The "on" pixels not belonging to the trajectories present in the image are filtered out by the SPTDS. After this, an image containing only trajectories and possibly some noise pixels, is obtained. The true trajectories have not, however, been identified yet. This is done by the Hough transform algorithm which also determines the coordinates of the last trajectory position.

The SPTDS has the potential of highly parallel implementation. A hardware implementation has not been developed yet. However, a hardware design for the Hough transform line detector has been developed. It consists in a parallel mapping from image to accumulator plane, plus a Hopfield network for peak detection. Both are VLSI implementable.

The complete subsystem has produced very satisfactory results in simulations using images with real background and simulated noise. The system is highly immune to noise.

2. RECOMMENDATIONS

1. **Motion/edge detection.** Further algorithm simplification for the purpose of improving processing time is recommended. A hardware design using DSP chips is necessary. With the

present algorithm processing time requirements this should be possible. If further development reduces processing time, implementation should be simpler. Preliminary studies seem to indicate that an ANN implementation is also feasible. This study is also recommended.

2. **Image segmentation.** The segmentation algorithm is easily implementable by means of a general purpose digital computer. A dedicated processor would enhance system speed. It should be decided whether it is more convenient to use the master control processor or a dedicated one for segmentation algorithm implementation.
3. **Centroid determination.** This subsystem is completely designed, including hardware circuit. What remains to be done is to determine whether it is possible to use a single VLSI chip or more than one will be necessary. The VLSI lay out can then be produced.
4. **Logarithmic spiral mapping.** This is straight forward and the special digital hardware implementation can be used.
5. **Multipixel target recognition.** The same observations made with respect to Centroid determination apply. The system has been designed using a hybrid hardware circuit. It remains to determine the possibility of single or multichip implementation and VLSI circuit lay out.
6. **Line-correlator target tracker.** Early in the research, a special purpose digital processor implementation was proposed. Later, a Hopfield neural network implementation was developed and simulated. It seems that present VLSI technology is insufficient for implementation of a 128x128 image. A lower resolution, 32x32, should be VLSI implementable. A VLSI lay out for key parts of the circuit has been developed and is presented in the *Engineering Drawings* report. It is recommended that further research be performed on the possibility of full resolution (128x128) implementation.
7. **Single pixel target detection.** The trajectory detection system hardware has to be studied in more detail. It is necessary, first of all, to determine whether present technology will allow implementation of the huge number of processing elements and associated connections required by the algorithm, both in its digital and its analog (ANN) parts. Once this study is completed, assuming that hardware implementation is feasible, a complete circuit design and VLSI lay out will be required.
8. **Hough transform single pixel target detection.** It has been shown that a multichip 128x128 resolution implementation, or a single pixel 32x32, is possible. What remains to be done is a VLSI lay out for both cases.

In addition to the above, it is necessary to consider that a master processor is needed to control signal flow, timing, etc. for the overall system.

SECTION XI

REFERENCES

[1] Holler, M., et al., "An electrically trainable artificial neural network (ETANN) with 1024 floating gate synapses," *Proc. of the IJCNN, Washington, DC, June 1989.*

[2] Fisher, W. A., et al., "A programmable analog neural network processor," *IEEE Trans. on Neural Networks,*, vol. 2, no. 2, March 1991.

[3] Canny, J., "A computational Approach to Edge Detection," *IEEE Trans. PAMI*, vol. PAMI 8, no. 6, November 1986.

[4] Korn, A. F., "Toward a Symbolic Representation of Intensity Changes in Images," *IEEE trans. PAMI*, vol. 10, no. 5, September 1988.

[5] Hsing, C., Inigo, R. M. and McVey, E. S., "Image Motion Detection and Estimation," *Joint SPIE-IECON Conf.*, Philadelphia, PA, November 5, 1989.

[[6] Burt, P., "Fast Filter Transforms for Image Processing," *Computer Graphics and Image Processing*, vol. 16, No. 1, 1981.

[7] Hopfield, J.J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Science, USA*, vol. 79, pp. 2554-2558, 1982.

[8] Hopfield, J.J., "Neurons with Graded Response Have Collective Computational Properties Like Those of 2-State Neurons", *Proceedings of the National Academy of Science, USA*, vol. 81, pp. 3088-3092, 1984.

[9] Hopfield, J.J. and Tank, D.W., "'Neural' Computation of Decisions in Optimization Problems," *Biological Cybernetics 52*, pp. 141-152, 1985.

[10] Hopfield, John J. and Tank, David W., "Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Transactions on Circuits and Systems*, Vol. CAS-33, No. 5, pp. 533-541, May 1986.

[11] Reid, M. B. et al., "Simultaneous Position, Scaling and Rotation Invariant Pattern Classification Using Third Order Neural Networks," *Int. Journal NNs-Research and Applications*, vol. 1, no. 3, pp. 154-159, July 1989.

[12] Lee, Y. C. and Doolen, G., "Machine Learning using a Higher Order Correlation Network," *Physica*, 22D, pp. 276-306, 1986.

[13] Rosenblatt, F., "Principles of Neurodynamics: Perceptions and the Theory of Brain Mechanisms," Spartan Books, 1961.

[14] Minsky, M. L. and Papert, S., *Perceptron*, MIT Press, Cambridge, MA, 1969.

[15] Marr, D., "A Theory for the Cerebral Cortex," *Proc. Royal Soc.*, London, B176, pp. 161-234, 1970.

[16] Kohonen, T. et al., "A Principle of Neural Associative Memory," *Neuroscience*, vol. 2, pp. 1065-1076, 1977.

[17] Kohonen et al., "Storage and : cessing of Information in Distributed Associative Memory Systems," in *Parallel Mode4ls of Asso. Mem.*, Hinton and Anderson, Ed. Chap 4., 1981.

[18] Aloimonos, J. and Basu, A.,"Shape and 3D Motion from Contour Without Point to Point Correspondence: General Principles," *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 518-527, 1986.

[19] Katanati,A., "Tracing Planar Surface Motion from Projection Without Knowing the Correspondence," *Computer Vision, Graphics and Image Processing, 21, 205-221, 1983.*

[20] Hopfield, J.J., and Tank, D.W., "'Neural' Computation of Decisions in Optimization Problems", *Biological Cybernetics 52*, pp.141-152, 1985.

[21] Jain, A.N., and Kring, D.B., "A Robust Hough Transform Technique", *VISION, SME*, pp.1-4, Fall 1987.